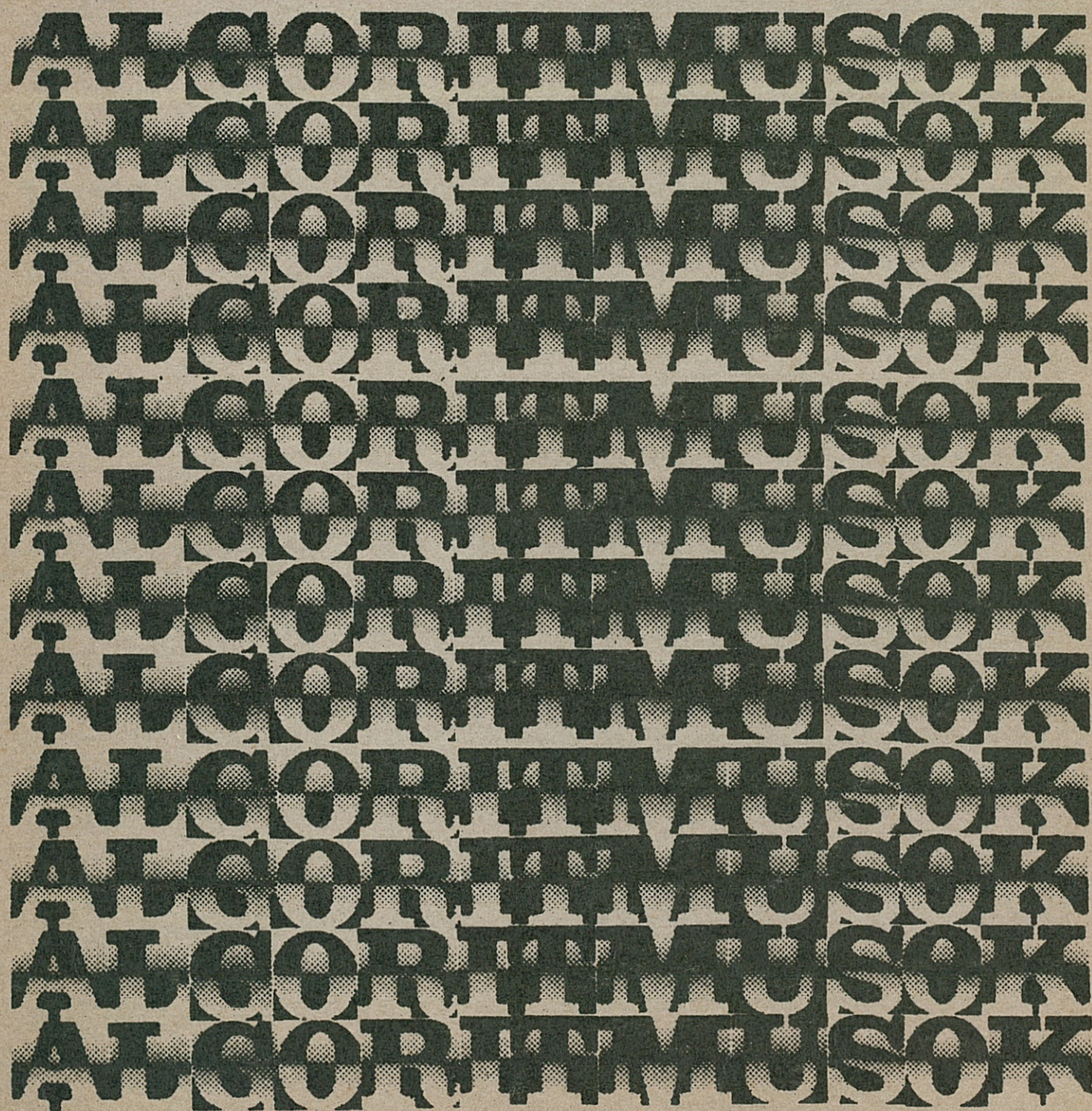


GÁCS – LOVÁSZ

ALGORITMUSOK



Gács Péter — Lovász László

Algoritmusok

Harmadik kiadás

Tankönyvkiadó, Budapest, 1989

EGYETEMI TANKÖNYV
kiadását a művelődési miniszter rendelte el

Bírálok:

Hajnal András egyet. tanár
Szűcs Ervin egyet. tanár

ETO: 511.1
512.1
519.1

ISBN 963 18 1618 4

© Dr. Gács Péter—Dr. Lovász László, 1978

Tartalomjegyzék

Bevezetés 9

1. fejezet.

Kombinatorikai algoritmusok 15

1.1. Szortírozás 17

1.1.1. Beszűrös 17

1.1.2. Összefésülés 18

1.1.3. Sorbarendezés 19

1.1.4. Átrendezés cserékkel 22

1.1.5. Legnagyobb és következő elemek 26

1.1.6. Mediánskeresés 27

1.2. A pakolási probléma 29

1.2.1. Próbálgató módszerek 31

- 1.2.2. Intervallumpakolás 33
- 1.2.3. Párosítások 35
- 1.3. Útkeresés 39
 - 1.3.1. Összefüggőség 39
 - 1.3.2. A legrövidebb út 41
 - 1.3.3. A leghosszabb út 44
- 1.4. Szuboptimális algoritmusok 47
 - 1.4.1. A hátizsákfeladat 47
 - 1.4.2. A fedési feladat 49
 - 1.4.3. A kromatikus szám 53
- 1.5. Gráfok diagnosztikája 54
 - 1.5.1. Összefüggőség 55
 - 1.5.2. Kieséses verseny és egy rokon feladat 57
 - 1.5.3. A probléma általánosítása 60

2. fejezet.

Kereső feladatok 63

- 2.1. A problémák problémája 64
 - 2.1.1. Ahol egy kismadár is segíthet 64
 - 2.1.2. A kielégíthetőség problémája — univerzális feladat 66
- 2.2. Nehéz kombinatorikai feladatok 70
 - 2.2.1. Közvetlen átfogalmazás 70
 - 2.2.2. A háromszín-probléma 71
 - 2.2.3. A pakolási probléma 73
- 2.3. Áttekintés 74

3. fejezet.

Algebrai algoritmusok 79

- 3.1. Gyorsabban is lehet szorozni 80
 - 3.1.1. Nagy számok szorzása 80
 - 3.1.2. Nagy mátrixok szorzása 82
 - 3.1.3. Polinomok szorzása 85
 - 3.1.4. A véges Fourier-transzformált 88
- 3.2. Osztás helyett szorzás 90
 - 3.2.1. Számok osztása 90
 - 3.2.2. Hatványsorok és polinomok osztása 92
 - 3.2.3. Az euklidészi algoritmus 97
 - 3.2.4. Mátrixinvertálás 104

- 3.3. Polinomok kiértékelése 106
 - 3.3.1. Pan alsó becslései 107
 - 3.3.2. Prekondicionálás 113
 - 3.3.3. Tabellázás és fordítottja, az interpoláció 117
- 3.4. Párhuzamos számolás 123
- 3.5. Moduláris algoritmusok 130
 - 3.5.1. A kínai maradéktétel 130
 - 3.5.2. Lineáris egyenletrendszer pontos megoldása 137

4. fejezet.

A kiszámíthatóság elméletéről 145

- 4.1. Az algoritmikus megoldhatóság 145
 - 4.1.1. Az algoritmus fogalma 146
 - 4.1.2. Az univerzális gép. A megállási feladat 148
 - 4.1.3. Egyszerű eldönthetetlen feladatok 150
- 4.2. A számolások terjedelméről 153
 - 4.2.1. A terjedelem mérése 156
 - 4.2.2. Optimális algoritmusok 158
 - 4.2.3. Alsó becslések 160

Feladatmegoldások 163

Bevezetés

Az algoritmus legáltalánosabb értelemben nem más, mint tervszerűség. Ha egy elvégzendő cselekvéssorozatot lépésről lépésre előre átgondolunk, megtervezünk, úgy is mondhatjuk, hogy algoritmust adunk egy bizonyos cél elérésére. Egy épület terve, vagy egy ételrecept — ilyen legtágabb értelemben vett algoritmus. Algoritmust követ a futballbíró, amikor mérkőzést vezet — hiszen előre el van tervezve, méghozzá nem is ő maga, hanem a játék szabályai mondják meg, hogy milyen szituációban mit kell ítélnie. Sok mindennapi tevékenységünk szigorúan véve nem algoritmikus. Megállhatunk közben töprengeni, vagy akár csak magunk elé meredni, hátha „beugrik” egy jó ötlet. Ha azonban sokszor kell őket végrehajtani, előbb-utóbb „algoritmizál” az ember. Gondoljunk például a csomagolásra valamilyen utazáshoz, vagy egy tárgy keresésére a lakásban. Egy tevékenységet csak akkor mondunk teljesen algoritmizáltnak, ha már pontosan leírtuk, elterveztük,

hogy milyen lépéseket fogunk végrehajtani (esetleg a korábbi lépések eredményétől függően); végrehajtás közben további módosításra vagy gondolkodásra nincs szükség.

Nemcsak fizikai tevékenységet, hanem gondolatit is lehet tervezni, „algoritmizálni”. A bolti pénztáros, ha visszaad, egy egyszerű algoritmust követ, melynek segítségével végül is eljuttatja a vásárlónak a visszajáró pénzt. Egy hivatal eljárási rendje, egy zene kottája, egy bridzs-licitrendszer — algoritmusok. Persze sok ezek közül nem elégíti ki a szigorúbb kikötéseket: ugyanabból a kottából ugyanazt a zenét sokféleképpen le lehet játszani, ugyanabból a lapból ugyanazzal a licitrendszerrel lehet kis szlemmet játszani és nagy szlemmet bukni. Mindezek a tevékenységek érdekességüket, sőt értelmüket vesztenék, ha teljesen algoritmizálva volnának.

A legjobb példákat a mindennapi életben használatos algoritmusokra a matematika szolgáltatja. Hogyan adunk (vagy szorzunk) össze két számot? Megtanultunk rá egy mindig működő, pontosan leírt, gépiesen végrehajtható eljárást, egy algoritmust. Ugyanez a helyzet az osztással, a gyökvonással, a háromszög területének kiszámításával, a százalékszámítással stb. Az általános iskolai matematikaanyag, de még részben a középiskolai is, főként algoritmusokat tanít az ilyen gyakran felvetődő feladatok megoldására.

Talán meglepő egy „Algoritmusok” c. könyv részéről, ha ezzel a gyakorlattal kapcsolatban kételyeknek ad hangot. Véleményünk azonban az, hogy az emberi szellemi tevékenység lényege nem algoritmusok végrehajtása, hanem algoritmusok alkotása. Természetes, hogy bizonyos alapvető algoritmusokat (összeadás, szorzás) mindenkinek ismernie kell. Azonban éppen az alapfeladatokat olyan sokszor, annyira változó körülmények között kell megoldanunk, hogy egy adott algoritmust — akár a legközönségesebbet, például a szorzását — legjobbnak, „egyedül üdvöztetőnek” deklarálni nem lehet. Éppen a szorzással kapcsolatban látni fogjuk, hogy nagyon sok jegyű számok szorzására van gyorsabb, jobb eljárás, mint amit az iskolában tanultunk! Ilyen sok jegyű számok szorzására ma még ritkán van szükség, de ahogy a számítások mennyisége növekszik — a hibák halmozódása miatt —, a pontosságot, tehát a jegyek számát is növelni kell.

Könyvünk célja sem az, hogy algoritmusok lexikonja legyen. Sok feladatra adunk algoritmust, némelyekre több különbözőt is, de ezek főként példaként szolgálnak. Fő célunk az, hogy az olvasó arról kapjon képet, hogyan is kell algoritmust keresni, tervezni egy-egy feladat megoldására.

Az algoritmus, most már a szűkebb, precíz matematikai értelemben véve, önálló matematikai létező. Egy algoritmus megvizsgálható sokféle szempontból (gyorsaság, tárigény stb.), két

algoritmus összehasonlítható, sőt olyan kérdések is felvethetők, hogy egy feladat megoldására alkalmazható összes lehetséges algoritmusok közül adott szempontból melyik a legjobb (vagy van-e egyáltalán legjobb). Vagy: létezik-e egyáltalán algoritmus egy-egy feladat megoldására? Az ilyen vizsgálatok még gyerekcipőben járnak, de ezek alkotják az algoritmusok elméletének legérdekesebb fejezeteit. E könyvben igyekezni fogunk az ilyen eredményekből is legalább ízelítőt adni.

Az előbb már említettük a számítógépeket, de részletesebben is kell szólnunk róluk, mivel az algoritmusok elméletében és gyakorlatában egyaránt forradalmi változást hoztak.

Bár „algoritmusok” (tervszerűség) mindennapi életünkben is lépten-nyomon előfordulnak, és szűkebb értelemben vett, matematikai algoritmusokat már a görögök is alkottak, az algoritmus fogalmának precíz megalkotására csak az 1930-as években került sor. Ekkor bizonyította be *Church*, hogy van olyan feladat, amely algoritmussal nem oldható meg. Az az algoritmusfogalom, melyet ennek az időszaknak a matematikai logikusai használtak, kellemtelenül bonyolult volt és egyáltalán nem szemléletes.

Ma már szinte mindenki hallott, olvasott vagy tanult arról, hogy hogyan működik egy elektronikus számítógép; egyre többen írtak már programot is. Így egyre több olyan ember van, akinek számára igen szemléletessé teszi az algoritmus fogalmát a következő kijelentés: algoritmus az olyan eljárás, amely számítógépre programozható. Ez persze nem matematikai definíció, de azzá tehető, ha a számítógép fogalmát pontosan leírjuk, és ezt fogjuk tenni könyvünk vége felé.

A számítógépek az algoritmusok alkalmazhatóságát sokszorosára növelték. Olyan méretű feladatoknál, melyek a számítógépek megjelenése előtt megoldhatók voltak, a teljes gépesítés nem okvetlenül volt előnyös, hiszen az algoritmus „lefuttatását”, a gépies számítási munkát is embernek kellett végeznie, és jelentősen növelhette a hatékonyságot, ha ez az ember is ügyeskedhetett, kihasználhatta menet közben az előre nem látható egyszerűsítési lehetőségeket. A gépek megjelenése megváltoztatta a helyzetet: sokkal nagyobb, bonyolultabb számítási, adatfeldolgozási feladatokat tudnak megoldani, mint az ember magában, de ehhez pontosan, egyértelműen megfogalmazott programra, algoritmusra van szükségük. Ezek az algoritmusok a gépek nélkül gyakorlatilag használhatatlanok, ezért korábban a megfelelő feladatok fel sem vetődtek. A gép így aztán olyan munkákat is átvállalhatott az embertől, melyekről az úgy érezte, hogy intelligenciája, nemformális ítélőképessége legjavát követelik tőle (pl. a piaci döntéseket).

Elégge elterjedt az a vélemény, hogy néhány döntő ponton az embert sohasem lehet gépekkel helyettesíteni. A kérdésnek

természetesen csak olyan feladatok esetén van értelme, amelyek a gépek számára egyáltalán megfogalmazhatók. Ha elfogadjuk, hogy e feladatok megoldására az emberek is algoritmust használnak, akkor a helyettesítésnek csak gyakorlati akadálya lehet: az, hogy mi ezeket az algoritmusokat nem ismerjük (csak használjuk: könnyebb megtanulni pingpongozni, mint leírni, hogyan kell jól pingpongozni), vagy nem rendelkezünk elvégzésükhöz szükséges megfelelő mennyiségű és struktúrájú gépi számolási kapacitással. Röviden: hogy az emberi agy a gépeknél minőségileg bonyolultabb módon dolgozik. Van olyan álláspont is, hogy az igazán magasrendű emberi tevékenységek nem algoritmizálhatók: eredendően nem formálisak. Akárhogyan vélekedünk is ezekről a kérdésekről, algoritmusokat gyártani a legkülönbébb célokra — kétségtelenül hasznos tevékenység, és nincs izgalmasabb dolog, mint megfelelő algoritmusok segítségével utolérni az embert valamilyen bonyolult funkciójában (pl. az előbeszéd áttétele írásba).

Az algoritmusok készítése iparrá lett. Általános rendeltetésű programok, a gépekkel együtt vagy külön, piacon vásárolhatók. Vannak cégek, melyek kizárólag programkészítéssel foglalkoznak megrendelésre.

Könyvek, folyóiratok folyamatosan közölnek algoritmusokat. A „Communications of the ACM” nevű folyóirat sok év óta egy-egy rendszer szerint számozza az általa közölt algoritmusokat: visszatér rájuk újabb számolási tapasztalatok, javított változatok megjelenése alkalmából. (Algoritmusgyűjteménye e fajtában egyedülállónak számít.)

Végül néhány szót a szükséges előismeretekről. Feltételezzük, hogy az olvasó tisztában van az algebra és kombinatorika alapfogalmaival. Egy-két helyen lineáris algebrai ismeretekre is építünk. Nem kívánunk jártasságot a számítógép-programozásban, de feltételezzük, hogy az olvasó ismeri a következő, egy számítógép működésének és programozásának megértésénél alapvető fogalmakat: központi és háttértár (memória), blokkdiagram.

A felhasznált algebrai, számelméleti és kombinatorikai ismeretanyag kifejtése megtalálható a következő könyvekben:

Birkhoff—Bartee: A modern algebra a számítógép-tudományban. Műszaki Könyvkiadó, 1974.

Busacker—Saaty: Véges gráfok és hálózatok. Műszaki Könyvkiadó, 1969.

W. Mayeda: Alkalmazott gráfelmélet. Műszaki Könyvkiadó, 1976.

Andrásfai Béla: Ismerkedés a gráfelmélettel. Tankönyvkiadó, 1971.

Rózsa Il.: Lineáris algebra műszaki alkalmazásokkal. 2. kiadás. Műszaki Könyvkiadó, 1976.

Anthony Ralston: Bevezetés a numerikus analízisbe. Műszaki Könyvkiadó, 1969.

Azok számára, akiknek érdeklődését sikerült felkeltenünk az algoritmuselemzés témaköre iránt, elsősorban

Trahtenbrot: Algoritmusok és absztrakt automaták. Műszaki Könyvkiadó, 1979., valamint D. Knuth — magyarul is megjelenés alatt álló könyvét ajánljuk:

Donald Knuth: A számítógép-programozás művészete (Műszaki Könyvkiadó).

- I. Alapvető algoritmusok
- II. Szeminumerikus algoritmusok
- III. Rendezés és keresés

1. fejezet

Kombinatorikai algoritmusok

A sakkjáték feltalálójának jutalmul felajánlotta a perzsa sah, hogy azt kérhet, amit akar. „Szerény leszek, ó, királyok királya” — válaszolta ő. „Tégy a sakktábla első mezejére egy búzaszemet, a másodikra kettőt, a harmadikra négyet, a negyedekre nyolcat és így tovább, minden soron következőre kétszer annyit, mint a megelőzőre. Csak ennyi búzát kérek!” Nagy volt a sah meglepetése, amikor kiderült, hogy nemcsak a búza nem fért volna el a táblán, de az idők kezdetétől nem termett még annyi búza ($2^{64} - 1$ szem), amennyit a feltaláló kért!

Egyszerű kombinatorikai feladatok könnyen vezetnek ilyen elképesztően nagy számokhoz. Ahhoz, hogy egyszerű kombinatorikai kérdésekre választ adhassunk, gyakran kellene ilyen elképesztően sok esetet megvizsgáljunk.

Sok kombinatorikai feladattal találkozunk rejtvényfejtés közben. Például: jussunk át egy labirintuson. Ha minden elágazásnál

minden lehetséges irányt végig akarunk próbálni, nagyon-nagyon sokáig kellene próbálkoznunk.

A gyakorlott rejtvényfejtő persze nem néz végig minden esetet. Ötletei vannak (pl.: a labirintus jobb feléről a bal felére csak két helyen lehet egyáltalán átjutni, tehát ezek valamelyikén feltétlenül át kell, hogy haladjunk), elképzelései, megérzései (erre jó lehet menni, mert sok lehetőség van a folytatásra), sőt megpróbálja kitalálni a rejtvénykészítő gondolatait. A szokásos rejtvényeknél ez a kellemes agymunka és az izzadságos, unalmas „esetszétválasztás” arányban vannak egymással. De kezdjük csak el növelni a labirintus méreteit! A megvizsgálandó esetek száma gyorsan nő, mint a búzaszemeké a sakktábla mezőin; hamarosan nemcsak hogy túlságosan hosszúvá és unalmassá, hanem megoldhatatlanul bonyolulttá válik a feladat.

Pedig ilyen „kombinatorikus”, tehát véges halmazokkal, struktúrákkal, „kombinálgatással” foglalkozó feladatok a gyakorlatban is igen fontosak. Hogyan állítsuk össze a vasúti menetrendet? Milyen sorrendben járja be a házakat a levélkézbesítő? Hogyan osszuk szét a termelési feladatokat a vállalatok között? Néhány vonat, ház, vállalat esetében a (bizonyos szempontból) optimális megoldás megkeresése akár érdekes rejtvényfejtés is lehet. De egy város vagy ország méreteiben?

Az ilyen kérdéseket a modern számítógépek tették megközelíthetővé. Ezek könnyen végigvizsgálják az esetek millióit, így mentesíthetik az embert a gépies, unalmas munkától.

Persze, a gép csak azt csinálja, amit „mondanak” neki. A rejtvényfejtéshez szükséges ötleteket az embernek kell kitalálni és a gépnek „megtanítani”. A feladat megoldásában segítséget nyújtó ötletek, egyértelmű és precíz utasításokként megfogalmazva: ez az algoritmus.

Kombinatorikus algoritmusokat nem is nagyon vizsgáltak módszeresen a számítógépek megjelenése előtt; más szóval, senki sem vette magának a fáradságot, hogy a „rejtvényfejtés” közben használt ötleteket összegyűjtse és tanulmányozza. Így aztán előállt az a helyzet, hogy még ma sem tudjuk teljes egészében megválaszolni az olyan egyszerű kérdéseket, mint „Hogyan jó n dolgot sorbarendezeni?” vagy „Hogyan jó kiválasztani felülről a 10-ediket egy listából?” (Míg pl. az égitestek pályaszámításairól, vagy igen bonyolult, folytonos, de nem differenciálható függvények létezéséről igen sokat tudunk.)

E fejezetben néhány ilyen alapkérdésre keresünk választ. Sok példánk a legtöbbet vizsgált kombinatorikus struktúrák, az ún. gráfok köréből való. A gráfok igen általános, az elmélet és gyakorlat majdnem minden területén előforduló struktúrák. (Bizonyos dolgok, melyek között egyfajta kapcsolat, reláció áll fenn: gráf.)

Ha az olvasó mégis túlzásba vittnek tartaná a gráfok szerepeltetését, legyen elnéző az egyik szerző e személyes szimpátiája iránt. Könyvünk célja, hogy az algoritmusok elméletének helyzetéről, erejéről, fejlettségéről, módszereiről képet adjon, s így jobb összehasonlítási alapot kapunk, ha példáinkat egy fogalom köré csoportosítjuk.

1.1. Szortírozás

1.1.1. Beszúrás

Elkészítettünk egy listát $n-1$ dologról (pl. egy névsort). Ezután kapunk egy n -ediket, és ezt a helyére akarjuk tenni. Ez úgy értendő, hogy a listán szereplő bármely dologgal összehasonlítva meg kell tudnunk mondani, hogy előtte vagy utána következik-e. Hogyan csináljuk ezt?

A legegyszerűbb eljárás végigszaladni a listán, amíg az első olyan dologhoz nem érünk, amelyet a beszúrandó már nem előz meg, ekkor a szóban forgó n -ediket ez elé tesszük. Ez az eljárás néha lehet, hogy a legjobb, de van egy hátránya: nagyon sok (a legrosszabb esetben n , általában $n/2$) összehasonlítást igényel. Ha a helyzet olyan, hogy két dolog összehasonlításához hosszabb számítást vagy kísérletet kell végeznünk, érdemes másképpen eljárni.

Egyszerűség kedvéért tegyük fel, hogy $n=2^k$ (ha n nem ilyen alakú, akkor fiktív dolgokat odaképzelve a sor végére n -et kiegészítjük 2-nek legközelebbi hatványává). Hasonlítsuk össze az új dolgot a középső elemmel; legyen pl. az eredmény az, hogy az új elem megelőzi a középsőt. Ekkor már csak az első $2^{k-1}-1$ elem közé kell beszúrni az újat. Ismét összehasonlítva ezek közül a középsővel, sikerült $2^{k-2}-1$ elem közé való beszúrásra redukálni a feladatot. Hasonlóan továbbmenve, $k-1$ összehasonlítás után „egy elem közé való beszúrásra” van csak szükség, amihez 1 összehasonlítás kell. Így a k lépésszám n -nel kifejezve: $k=\log_2 n$, vagy ha n nem volt 2-nek hatványa, akkor k a $\log_2 n$ -re következő legelső egész szám, melyet $\lceil \log_2 n \rceil$ -nel jelölünk.

Az összehasonlítások száma ezzel igen lecsökkent az első eljáráshoz képest, pl. 1000 dolog esetén 1000 helyett 10-re. Ennek fejében a programozás bonyolultabbá vált. Megmutatjuk most, hogy az összehasonlítások számát nem lehet tovább csökkenteni. Ez a meggondolás igen egyszerű példája lesz egy gyakran használható módszernek: hogyan követhetjük nyomon az egyes lépések után rendelkezésiünkre álló információ mennyiségét.

Kezdetben semmit sem tudunk arról, hogy hová kell beszúrni az új elemet. Tegyük fel, hogy elvégeztünk egy összehasonlítást a j -edik elemmel; mondjuk $j \leq \frac{n+1}{2}$. Ekkor vagy az adódik, hogy ez előtt, vagy hogy ez után van az új. Adódhat az, hogy utána; ekkor még $n - j \left(> \frac{n}{2} - 1 \right)$ elem közé kell az újat beszúrni, amelynek ezen elemekhez való viszonyáról semmit nem tudunk, így közöttük bárhol lehet. Hasonlóan továbbmenve látjuk, hogy l lépés után, ha nincs szerencsénk, még mindig van legalább $n/2^l - 1$ egymás utáni elem, amelyeknek az újhoz való viszonyáról nem tudunk semmit. Ez még $l = \lceil \log_2 n \rceil - 1$ esetén is legalább egy elem, ennél tehát több lépés kell.

1.1. feladat: 16 látszatra egyforma pénzdarab között egyik hamis, könnyebb a többinél. Egy egykarú, skálás mérlegen közülük akárhány érme összsúlyát meg tudjuk határozni, és ebből megállapítani, hogy közöttük van-e a hamis. Keressük meg a hamis pénzt minimális számú méréssel.

Hasonlítsuk össze feladatunkat a beszúrási feladattal.

1.2. feladat: Keressük meg 27 pénzdarab közül a hamisat, ha egy kétkarú mérlegünk van, súlyok nélkül!

1.1.2. Összefésülés

Rokon probléma a következő: van két, n hosszúságú listánk; egyesítsük őket! Ismét az a helyzet, hogy a listán szereplő dolgoknak van egy *jól meghatározott sorrendje*, és *két dolgot össze tudunk hasonlítani*. Szeretnénk kevés összehasonlítással megvalósítani az összefésülést.

Legyen a két lista $(x_1, \dots, x_n)$ és $(y_1, \dots, y_n)$. Eljárásunk a következő. Hasonlítsuk össze az y_1 elemet rendre az $x_1, x_2, \dots$ elemekkel, míg meg nem találjuk az első olyat, mondjuk x_2 -t, melyet megelőz; ekkor tegyük be y_1 -et x_2 elé. Hasonlítsuk össze ezután y_2 -t az $x_2, x_3, \dots$ elemekkel, amíg egy olyat nem találunk, melyet y_2 megelőz; tegyük be y_2 -t ez elé stb.

Megjegyezzük, hogy az eljárás kellemetlen tulajdonsága, hogy az x_i elemeket sokszor meg kell mozgatni. Ezen segíthetünk, ha „plusz” tárat használunk. Ekkor az eljárás így írható le: Hasonlítsuk össze a két első elemet. Írjuk le azt, amelyik megelőzi a másikat egy külön helyre, és egyben töröljük is az eredeti helyéről. Ismételjük ezt, míg valamelyik lista el nem fog, a másik lista maradékát írjuk át az új lista végére. Ezt az eljárást alkalmazva, minden elemet csak egyszer mozdítunk meg.

1.3. feladat: Gondoljuk át, hogy a második eljárás az elsőtől csak az adatok más elhelyezésében különbözik.

Hány összehasonlítást végzünk így? Minden összehasonlításnál egy elem kerül az új listára; az utolsó, a $2n$ -edik elemhez már nem is kell összehasonlítás. Így legfeljebb $2n - 1$ összehasonlításra volt szükség.

Könnyű megmutatni, hogy $2n - 2$ összehasonlítás nem mindig elegendő. Tegyük fel, hogy végső sorrendnek $(x_1, y_1, x_2, y_2, \dots, x_n, y_n)$ adódik. Ekkor itt bármely két szomszédos elem összehasonlítását el kellett, hogy végezzük az eljárás során, mert más összehasonlításokból a sorrendjükre következtetni nem lehet.

1.1.3. Sorbarendezés

Gyakori feladat, hogy n adott dolgot valamilyen szempont szerint sorba kell raknunk. Pl. n nevet névsorba, vagy n adatot nagyság szerint. Lehetséges, hogy a dolgoknak valamilyen „értéke” vagy „súlya” van, és pl. érték szerint növekvő sorrendbe akarjuk rakni őket. De lehet, hogy — mint pl. a névsor esetén — ilyen érték nincsen. Azt fogjuk ezért feltételezni, hogy bármely két adott dolgot tekintve, meg tudjuk mondani, melyik előzze meg a másikat a sorrendben. Ezt röviden a két dolog összehasonlításának fogjuk nevezni.

(I) Első és legtermészetesebb módja a feladat megoldásának az, hogy először a dolgokat valahogyan sorbarakjuk, majd megnézzük, hogy az első és második, a második és harmadik, ..., $(n - 1)$ -edik és n -edik elem sorrendje jó-e. Ha mindez jó, akkor az elemek máris megfelelő sorrendben vannak. Ha találunk két szomszédos elemet, melyek sorrendje nem megfelelő, ezeket felcseréljük. Sajnos, lehet, hogy ezzel egy másik szomszédpár sorrendje elromlik. Pl. ha három egymás utáni helyen álló szám:

...15 22 13...,

akkor a 22 és 13 felcserélése után a 15-nek és rákövetkezőjének sorrendje fordul rosszra. De újra és újra kezdve az eljárást, véges számú lépés után eljutunk a jó sorrendhez. Megmutatható ugyanis, hogy *a rossz sorrendben levő párok száma minden cserével csökken*, így előbb-utóbb elfogy.

1.4. feladat: Kézenfekvő, hogy az a legrosszabb eset, ha a kiindulásul vett sorrend éppen fordítottja a kívántnak. Pl. a számok csökkenő sorrendben vannak, és mi növekedően akarjuk őket rendezni. Ekkor a fent

leírt eljárás $\binom{n}{2}$ cserét igényel. Bizonyítandó, hogy általában legfeljebb $\binom{n}{2}$ csere kell, és elegendő legfeljebb $\binom{n}{2}$ szomszédos párt összehasonlítani.

Lehet-e jobb eljárást találni a sorbarakásra?

Ez sok mindentől függ. A fenti eljárás könnyen programozható, nem igényel több tárhelyet, mint amennyi az adatok elhelyezéséhez kell, és nem túl sok $\left(\text{legfeljebb } \binom{n}{2}\right)$ egyszerű műveletet igényel.

Hogy szükség van-e más eljárásra, azt az határozza meg, mennyire egyszerűek ezek a műveletek. Két adott „dolog” összehasonlítása ugyanis esetleg bonyolult kísérlet vagy számítás elvégzését igényli. Még ha ilyenre nincs is szükség, de az összehasonlítás végett a gép adatszalaijához vagy lassabb tárához kell fordulnunk, előfordulhat, hogy két dolog összehasonlítása sokkal, de sokkal drágább (időigényesebb) dolog, mint pl. a dolgokat átrendezni vagy hosszabb programot írni. Így aztán felvetődik a kérdés: nem lehetne-e olyan eljárást adni, mely *kevesebb összehasonlítással* (esetleg több egyéb jellegű lépés, vagy nagyobb tárigény árán) rendezi sorba az elemeket?

(II) Képzeljük el, hogy a dolgok közül $(n-1)$ -et már sorbarendeztünk és most kapjuk az n -ediket (mint amikor valaki egyenként szedi fel a kártyáit). Az n -edik dolgot kell „beszúrni” valahová. Ez, mint tudjuk, $\lceil \log_2 n \rceil$ lépésben megtehető. Az első $n-1$ dolgot ugyanilyen eljárással rendeztük sorba, utoljára $\lceil \log_2 (n-1) \rceil$ összehasonlítás árán szúrva be az $(n-1)$ -ediket az első $n-2$ közé. Hasonlóan továbbmenve látjuk, hogy összesen

$$\lceil \log_2 n \rceil + \dots + \lceil \log_2 3 \rceil + \lceil \log_2 2 \rceil < n \lceil \log_2 n \rceil$$

összehasonlításra van szükség.

1.5. feladat: Igazoljuk, hogy

$$\lceil \log_2 n \rceil + \dots + \lceil \log_2 2 \rceil = n \lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1.$$

Ez az eljárás lényegesen kevesebb összehasonlítást igényel, mint az előző; pl. $n=1000$ esetén 499 500 helyett csak 9001-et. Kellemetlen tulajdonsága viszont, hogy minden újabb elem beszúrásakor, tehát n -szer meg kell bolygatni a már elhelyezett elemek egy részét (helyet kell csinálni a beszúrt elemnek).

(III) Rendezzük sorba az elemek első felét és a második felét, majd fésüljük össze a kapott két sorozatot.

1.6. feladat: Igazoljuk, hogy ennek az eljárásnak a lépésszáma azonos a (II) eljárásával.

Mivel összefésülést csak $\lceil \log_2 n \rceil$ -szer kellett végrehajtanunk, a (II) eljárással szemben egy elemet csak $\lceil \log_2 n \rceil$ -szer kell megbolygatnunk. Ezzel szemben itt pótlólagos tárra van szükség az összefésülés elvégzéséhez.

Lehet-e ennél „jobb”, vagyis kevesebb összehasonlítással dolgozó algoritmust adni? Megmutatjuk, hogy lényegesen jobbat nem. Tegyük fel, hogy van egy eljárásunk, mely megadja a sorrendet. Nézzük meg, hány összehasonlításra van szükség. Elvégett gondoljuk meg, hogy mit tudunk, mennyi *információnk* van a sorrendről az egyes összehasonlítások után. Mielőtt nekikezdenénk, az n dolognak bármely sorrendje elképzelhető számunkra, így $n!$ lehetőség van. Az összehasonlítások sorozatával azt szeretnénk elérni, hogy egy híján minden lehetőség ki legyen zárva. Az első összehasonlítás az $n!$ esetnek pontosan a felét zárja ki. A későbbi összehasonlításoknál ez már nem okvetlenül áll; de kedvezőtlen esetben a lehetőségeknek legfeljebb a fele zárható ki. Az összehasonlításokkor ugyanis az addig még szóba jövő esetek két osztályra bomlanak aszerint, hogy mi az összehasonlítás eredménye. Nem tudjuk, mekkora ez a két osztály, de ha az összehasonlítás után a nagyobbikba tartozó sorrendek maradnak, akkor a szóba jövő esetek száma kevesebb, mint felével csökkent.

Így tehát k összehasonlítás elvégzése után előfordulhat (azaz akárhogy is végezzük az összehasonlításokat, van olyan eset, amikor előfordul), hogy még legalább $n!/2^k$ sorrend szóba jöhet, azaz az eddigi összehasonlításokkal összhangban van. Ha eljárásunk N összehasonlítással biztosan célba ér, azaz N összehasonlítás után már biztosan csak egy sorrend lehetséges, akkor tehát fenn kell álljon, hogy

$$n!/2^N \leq 1,$$

azaz

$$N \geq \log_2 (n!) = \log_2 n + \log_2 (n-1) + \dots + \log_2 2.$$

Ez azt mutatja, hogy az adott eljárásban szereplő összehasonlítások száma lényegesen már nem volna csökkenthető. Foglaljuk össze az eddigieket:

- **Tétel:** *Bármely algoritmus, amely n dolgot sorbarendezi, legalább $\log_2 (n!)$ összehasonlítást kell hogy tegyen. Van olyan algoritmus, mely $\log_2 (n!) + n - 1$ összehasonlítással sorba tudja rakni az n dolgot.*

M e g j e g y z é s e k:

1. A fenti bizonyításnak arra vonatkozóan, hogy minden sorbarendező algoritmusnak legalább $n \log_2 n$ összehasonlítást kell

tennie, információelméleti háttére van. Az információelméletből ismeretes ugyanis, hogy N tárgy közül egynek a kiválasztásához (ha mindegyik egyformán szóba jöhet) $\log_2 N$ „bit” (egység) információ kell. Esetünkben egy összehasonlítás egy „bit” információt szolgáltat. Hasonló feladatok a „keresési problémák” (pl. több érme közül a hamis kiválasztása mérlegeléssel). Ezekre itt most nem térünk ki. Érdekes összevetni e feladatokat a közkedvelt barkochba játékkal is.

2. Az általunk megadott algoritmusok csak kicsit hosszabbak, mint a bármely algoritmusra érvényes $\log_2 (n!)$ alsó becslés. A két algoritmus kombinációjával Ford és Johnson még egy kicsit csökkenteni tudta a lépésszámot, de — nagy n mellett — még mindig van egy kis különbség az alsó becslés és algoritmusuk lépésszáma között. A pontos helyzet nem ismeretes (de ennek persze csak elméleti érdekessége van).

3. A fenti tétel állítása a *legrosszabb esetre* érvényes: minden sorbarendező algoritmushoz vannak olyan bemenő adatok, hogy ezeken az algoritmusnak $\log_2 (n!)$ darab összehasonlítást kell tennie. Kérdezhetnénk azonban, nincs-e olyan algoritmus, ami átlagban — ha az összes sorrendet egyenlően valószínűnek tekintjük — kevesebb összehasonlítást végez. Információelméleti gondolatmenettel megmutatható, hogy nincs.

Vannak olyan algoritmusok, amelyek a legrosszabb esetben $\frac{n(n-1)}{2}$ összehasonlítást követelnek, átlagban pedig csak $c \cdot \log_2 (n!)$ darabot. Az átlagos lépésszám meghatározása sokszor fejlett matematikai technikát igényel. Az 1.1.4. pont példája a legegyszerűbbek közé tartozik.

1.1.4.

Átrendezés cserékkel

A most következő példa elsősorban azt illusztrálja, hogy lehet egy algoritmus *átlagos lépésszámát* kiszámolni.

A feladat ugyanaz, mint az előbb: adott n különböző szám, $K_1, K_2, \dots, K_n$, melyeket növekvő sorrendbe kell átrendezni.

A következő algoritmust használjuk.

1. *fázis* (a K_1 szám a helyére kerül):

1. *csere*: A K_1 számot összehasonlítjuk K_n -nel. Ha $K_n > K_1$, akkor továbblépünk, és K_{n-1} -gyel hasonlítunk össze. Ha $K_{n-1} > K_1$, megint továbblépünk stb. Ha pl. $K_1 > K_{n-2}$, akkor K_1 -et kicseréljük K_{n-2} -vel. Az új sorozat (függőleges vonallal vágtuk

le a sorozat két végéről azokat a részeket, amelyekkel K_1 -et már nem kell hasonlítani):

$$K_{n-2}|K_2K_3 \dots K_{n-3}K_1|K_{n-1}K_n.$$

2. *csere*: A csere után K_1 -et a *legelső* olyan számmal hasonlítjuk össze, amivel még nem hasonlítottuk: példánkban K_2 -vel. Ha $K_1 > K_2$, akkor továbblépünk és K_1 -et K_3 -mal hasonlítjuk össze stb. Ha pl. $K_5 > K_1$, akkor K_1 -et kicseréljük K_5 -tel. Az új sorozat:

$$K_{n-2}K_2K_3K_4|K_1K_6 \dots K_{n-3}|K_5K_{n-1}K_n.$$

3. *csere*: Most K_1 -et megint a *legutolsó* olyan számmal hasonlítjuk össze, amivel még eddig nem volt összehasonlítva és lefelé haladunk, amíg cserére nem kerül sor.

4. *csere*: A csere után K_1 -et a *legelső* új számmal hasonlítjuk össze stb.

Végül K_1 -et már a sorozat minden elemével összehasonlítottuk és a nála nagyobbak mind mögé, a nála kisebbek pedig elé kerültek. Ha pl. K_1 az i -edik helyre került, a következő kép alakul ki:

$$L_1, L_2, \dots, L_{i-1}, K_i, M_1, M_2, \dots, M_{n-i},$$

ahol már tudjuk hogy

$$L_1, L_2, \dots$$

kisebbségek, mint K_i , és

$$M_1, M_2, \dots$$

nagyobbak, mint K_i .

Ezzel lezárult az algoritmus 1. fázisa.

Többi fázis: Ami ezután hátravan, az külön az $L_1, L_2, \dots, L_{i-1}$, és az $M_1, \dots, M_{n-i}$ sorozatok rendezése ugyanezzel a módszerrel.

Példa: A sorozat legyen

$$503, 87, 512, 61, 908, 170, 275.$$

1. *fázis* (503 helyre tétele):

$$1. \text{ csere: } 275, 87, 512, 61, 908, 170, 897, 503$$

$$2. \text{ csere: } 275, 87 | 503, 61, 908, 170, 897 | 512$$

$$3. \text{ csere: } 275, 87, 170 | 61, 908, 503 | 897, 512$$

$$4. \text{ csere: } 275, 87, 170, 61 | 503 | 908, 897, 512$$

- 2.1. *fázis* (275 helyre tétele):

$$1. \text{ csere: } 61 | 87, 170, 275$$

- 2.2. *fázis* (908 helyre tétele):

$$1. \text{ csere: } 512 | 897, 908$$

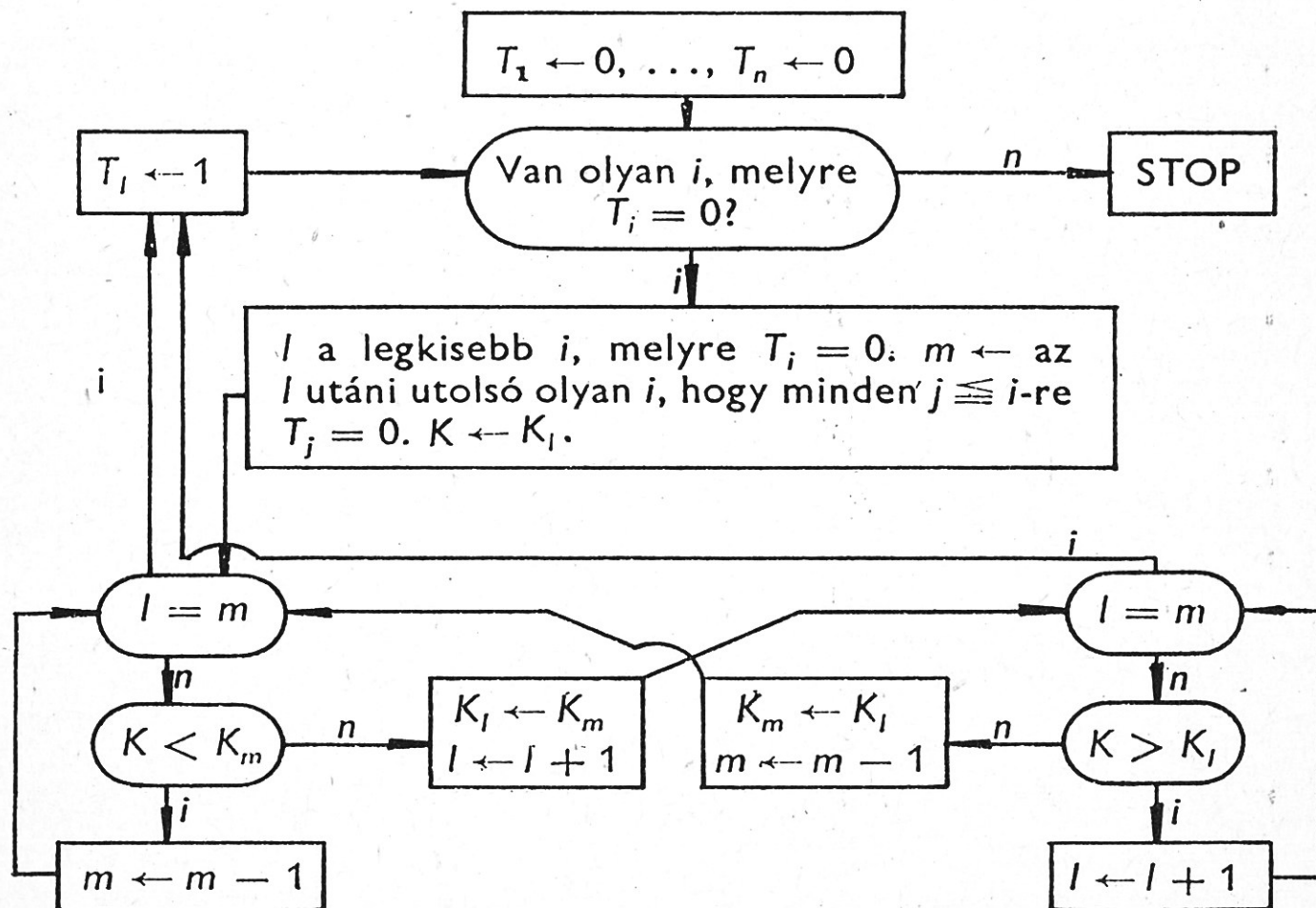
Eredeti formájában ez az algoritmus rekurzív, azaz önmagára hivatkozást tartalmaz. Ilyen algoritmusra nem lehet blokkdiagramot írni. Az ábra blokkdiagramja a rekurziót iterációvá oldja fel.

Bemenetkor: $K_1, \dots, K_n$ különböző számok.

Kimenet: $K_1, \dots, K_n$: ugyanezek a számok jó sorrendben.

Segédváltozók: $T_1, \dots, T_n$, 0 vagy 1 értékeket vehetnek fel. l, m egész számok 1 és n között. K a $K_1, \dots, K_n$ számok valamelyike.

Megjegyzés: Az eljárás folyamán $T_i = 1$, ha a K_i szám már nagyság szerint is az i -edik.



Ennek az algoritmusnak sok előnye van, pl. hogy az egész 1. fázisban ugyanazt a K_i számot kellett a többivel összehasonlítani. Ezért érdemes kiszámítani, hogy az algoritmus átlagban hány összehasonlítást igényel? Tegyük fel, hogy a $K_1, K_2, \dots, K_n$ számok minden lehetséges nagysági sorrendje egyformán valószínű. Az első fázisban K_1 -et az összes többi számmal össze kell hasonlítani, ez $n-1$ összehasonlítás. Könnyű belátni, hogy ha pl. az első fázis után az eredmény

$$L_1, L_2, \dots, L_{i-1}, K_1, M_1, \dots, M_{n-i},$$

akkor az $L_1, \dots, L_{i-1}$ számok minden lehetséges nagysági sor-

rendje is egyformán valószínű; az $M_1, \dots, M_{n-i}$ számoké hasonlóan. Jelöljük az n szám sorbarendezéséhez szükséges összehasonlítások átlagos számát $F(n)$ -nel! Ekkor, feltéve, hogy K_1 az i -edik helyre került, hátravan még átlagban $F(i-1) + F(n-i)$ összehasonlítás. Mivel pedig K_1 egyenlő valószínűséggel kerülhet bármelyik helyre, az összehasonlítások átlagos számára végül a következő rekurzív formulát kapjuk:

$$F(0) = 0, \quad F(1) = 0,$$

$$F(n) = n - 1 + \frac{1}{n} [F(0) + F(n-1) + F(1) + F(n-2) + F(2) + \dots + F(n-1) + F(0)].$$

Egyszerűsítve:

$$nF(n) = n(n-1) + 2[F(0) + F(1) + F(2) + \dots + F(n-1)]. \quad (1)$$

Írjuk fel ugyanezt az egyenlőséget $n+1$ -re:

$$(n+1)F(n+1) = (n+1)n + 2[F(0) + \dots + F(n)]. \quad (2)$$

Vonjuk ki (1)-ből (2)-t:

$$(n+1)F(n+1) - nF(n) = 2n + 2F(n).$$

Átalakítva:

$$(n+1)F(n+1) = 2n + (n+2)F(n),$$

$$\frac{F(n+1)}{(n+2)} - \frac{F(n)}{(n+1)} = \frac{2n}{(n+1)(n+2)}.$$

Adjuk össze a fenti egyenlőséget $n=1, 2, \dots, (k-1)$ -re:

$$\frac{F(k)}{k+1} = \frac{2 \cdot 1}{2 \cdot 3} + \frac{2 \cdot 2}{3 \cdot 4} + \frac{2 \cdot 3}{4 \cdot 5} + \dots + \frac{2(k-1)}{k(k+1)}.$$

Az olvasó egy kis számolással ellenőrizheti, hogy innen

$$F(n) = 2(n+1) \left(H_n - 2 + \frac{2}{n+1} \right),$$

ahol

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \sim \ln n.$$

Ezért a következő *aszimptotikus egyenlőséget* kaptuk az átlagos lépésszámmra: $F(n) \sim 2n \ln n$.

Érdekes megjegyezni, hogy a szóban forgó algoritmus a legrosszabb esetben az összes számpárt összehasonlítja, tehát $\frac{n(n-1)}{2}$ lépést dolgozik. Paradox módon ilyen legrosszabb eset az, ha a $K_1, K_2, \dots, K_n$ számok eleve jó sorrendben vannak, azaz $K_1 < K_2 < \dots < K_n$.

1.1.5. Legnagyobb és következő elemek

Maradjunk továbbra is annál a példánál, hogy n dolog van adva, melyeknek van egy sorrendje, de ezt nem ismerjük, és páronkénti összehasonlításokat tehetünk. Hány összehasonlítást kell tennünk, hogy a legnagyobb elemet megtaláljuk?

A válasz igen egyszerű. Többféle eljárás is adható, amely $n-1$ összehasonlítással célhoz ér. Tegyük fel, hogy n pingpongversenyzőről van szó, és a lehető legkevesebb mérkőzés lejátszása árán szeretnénk eldönteni, melyikük a legjobb. Feltesszük, hogy a versenyzők erőssorrendje egyértelmű, tehát két versenyző közül biztosan a jobbik győz. Ekkor kétfajta verseny is rendezhető.

(a) *Kihívásos verseny.* Valaki kikiáltja magát bajnoknak. A bajnokot bárki kihívhatja. A mérkőzés vesztese a további versenyekből kiesik, a győztest ismét bárki kihívhatja és így tovább. Mikor már nincs, aki a győztest kihívja, megvan a legjobb.

(b) *Kieséses bajnokság,* a közismert módon.

1.7. feladat: Bizonyítsuk be, hogy mindkét esetben pontosan $n-1$ játszmára van szükség a bajnokság lebonyolításához.

Mármost $n-1$ játszmára feltétlenül szükség van; hiszen ahhoz, hogy valakiről kijelenthessük, hogy ő a legjobb játékos, minden más játékos valakitől vereséget kell, hogy szenvedjen, ami legalább $n-1$ játszmát jelent.

Vizsgáljuk meg most a második legnagyobb elem kiválasztásának lehetőségét. Képzeljük el, hogy lezajlik egy kieséses bajnokság. Ekkor a győztes személye nem kétséges; az azonban már vitatható, hogy a döntő másik résztvevője a második legjobb. Lehet pl. hogy már az első fordulóban a két legerősebb versenyzőt sorsolták össze, s így a második legjobb már itt kiesett. Bárki, akit a győztes ütött el a továbbjutástól, igényt tarthat a „második legjobb” címre (más már nyilván nem). Ha az ilyenek száma m , akkor $m-1$ további mérkőzést kell játszanunk ahhoz, hogy közülük a legjobbat kiválasszuk. Így a szükséges mérkőzések száma

$$(n-1) + (m-1).$$

Mennyi m ? Legfeljebb annyi, ahány forduló volt; ez pedig, mivel a versenyben maradó játékosok száma mindig feleződik, $\lceil \log_2 n \rceil$. Így a második legjobb kiválasztásához

$$n + \lceil \log_2 n \rceil - 2$$

mérkőzésre van szükség. Más szóval, n tárgy közül, ennyi összehasonlítást végezve, meg tudjuk adni a második legnagyobbbat.

Megmutatható, hogy ennél kevesebb összehasonlítás (a legrosszabb esetben) nem elegendő; erre itt nem térünk ki.

1.1.6. Mediánskeresés

Tekintettel arra, hogy a második legnagyobb kiválasztása nehezebb probléma, mint a legnagyobbé, és ez a nehézség a harmadik stb. legnagyobb esetén tovább fokozódik, azt várták, hogy a „közepe tájáról” választani egy elemet (pl. $n = 2m + 1$ esetén az $(m + 1)$ -ediket, a „mediánt”), már nem lehet lényegesen ügyesebben, mint az összes elem sorbarendezeésével. Meglepő volt ezért *Blum, Floyd, Tarjan, Pratt* és *Rivest* azon eredménye, hogy bármely $1 \leq t \leq n$ esetén ki lehet választani a t -edik elemet n -nel arányos lépésszámú algoritmussal. Ezt az algoritmust fogjuk most ismertetni. Az algoritmus maximális lépésszámát $1 \leq t \leq n$ esetén $L(n)$ -nel jelöljük; ezt majd később fogjuk meghatározni.

Legyen x_0 a tekintett halmaz egy eleme; később x_0 kiválasztására még visszatérünk. Hasonlítsuk össze x_0 -t a halmaz többi elemével; ez $n - 1$ összehasonlítás. Legyen S_1 , ill. S_2 az x_0 -nál nagyobb, ill. kisebb elemek halmaza; legyen x_0 a k -adik elem, vagyis $|S_1| = k - 1$.

Ha $k = t$, akkor készen vagyunk. Tegyük fel, hogy $k > t$. Ekkor az S_1 halmazból kell a t -edik elemet kiválasztani, amit $L(k - 1)$ összehasonlítás árán meg tudunk tenni. Ha $k < t$, akkor viszont az S_2 halmazból kell a $(t - k)$ -adikat kiválasztani, amihez $L(n - k)$ összehasonlítás elég. Így legfeljebb

$$(n - 1) + \max [L(k - 1), L(n - k)] \quad (3)$$

összehasonlítás árán meg tudjuk határozni a t -edik legnagyobb elemet.

Ha itt k vagy $n - k$ közel van n -hez, akkor ez a korlát nyilván nem lesz valami jó. Azt kellene ezért elérni, hogy k is, $n - k$ is, mondjuk, valahol $n/2$ körül legyen.

Ezt x_0 alkalmas választásával szeretnénk elérni. Persze ehhez újabb lépésekre lesz szükségünk, de reméljük, nem túl sokra.

Osszuk be az elemeket 5-ös csoportokba (ilyenből $\frac{n}{5}$ lesz; egyszerűség kedvéért tegyük fel, hogy n 5-tel osztható), majd válasszuk ki mindegyik 5-ös csoport mediánját.

1.8. feladat: Igazoljuk, hogy 5 elem mediánját ki lehet választani 6 összehasonlítás árán, de kevesebb nem.

Ehhez tehát $\frac{6n}{5}$ összehasonlítás kellett. Most válasszuk ki a mediánok mediánját (tegyük fel ismét az egyszerűség kedvéért, hogy $n/5$ páratlan), és legyen ez x_0 . Ehhez legfeljebb $L\left(\frac{n}{5}\right)$ összehasonlításra van szükség.

Ekkor x_0 -nál biztosan kisebb azon 5-ös csoportok 3–3 alsó eleme, melyek mediánja x_0 -nál kisebb; továbbá 2 elem a saját csoportjában. Ezek száma tehát legalább $3 \cdot \frac{1}{2} \left(\frac{n}{5} + 1 \right) > \frac{3}{10} n$, és így x_0 -nál nagyobb elem kevesebb, mint $\frac{7}{10} n$ van. Tehát $k < \frac{7}{10} n$ és hasonlóan, $n - k > \frac{7}{10} n$. Ezért a továbbiakban (3) szerint legfeljebb

$$n + L\left(\frac{7}{10} n\right)$$

összehasonlítás kell. Összegezve, a szükséges összehasonlítások száma:

$$L(n) \leq \frac{6}{5} n + L\left(\frac{n}{5}\right) + n + L\left(\frac{7}{10} n\right).$$

Ez az egyenlőtlenség lehetőséget ad az

$$L(n) \leq 22n \tag{4}$$

egyenlőtlenség teljes indukciós bizonyítására. Kis n értékre ($n \leq 5$) ez triviálisan igaz. Mármint ha $n < N$ -re igaz, akkor (4) alapján:

$$L(N) \leq \frac{6}{5} N + \frac{22}{5} N + N + \frac{7}{10} 22N = 22N.$$

A 22-es együttható leszorítható kb. 6-ra, ennek részleteire azonban már nem térünk ki.

1.2.

A pakolási probléma

1. példa: n dolgot kellene „bepakolni” egy ládába. Vannak azonban olyan párok, melyek bizonyos okok miatt nem férnek össze (erről teljes információnk van). Válasszunk ki maximális számú tárgyat úgy, hogy ezeket betehessük a ládába, vagyis közülük bármely kettő összeférjen.
2. példa: Válasszunk ki n ember közül a lehető legtöbbet úgy, hogy mindegyik ismerje a másikat!
3. példa: Egy teherautó A városból B városba megy. Útja mentén vannak az $A_1, \dots, A_k, B_1, \dots, B_k$ helyek (nem okvetlenül ilyen sorrendben). Minden A_i helyről a megfelelő B_i helyre el kellene szállítania valamit; egyszerre csak egy dolgot szállíthat. Mely szállításokat vállalja el, hogy a lehető legtöbb kívánságnak eleget tudjon tenni?
4. példa: Egy bőrből vágjuk ki a lehető legtöbb (adott alakú) cipőfelsőrészt!

Ezekben a problémákban sok közös van: ki akarunk választani maximálisan sokat egy összességből, olyan feltételek mellett, hogy bizonyos párok egyszerre nem választhatók ki. A 4. példában: ki akarunk választani maximálisan sokat az adott cipőfelsőrészalak összes lehetséges helyzetei közül úgy, hogy semelyik kettő se fedje át egymást. (Ez a példa abban különbözik a többitől, hogy benne a tekintendő összesség végtelen. Ha azonban eljárást szeretnénk nyerni az optimális elhelyezés meghatározására, a végtelen sok elhelyezkedés helyett úgyis csak egy elég finoman felosztott, de véges „listát” veszünk figyelembe.)

A viszonyokat igen szemléletesen írhatjuk le a következőképpen. Minden dolgot egy ponttal ábrázolunk. Ha két dolog összeférhetetlen, kössük össze őket egy szakasszal. Az összekötő szakaszokat *éleknek* fogjuk nevezni. Feladatunk az, hogy minél több pontot válasszunk ki úgy, hogy közülük semelyik kettőt ne kösse össze él. Az ilyen ponthalmazt röviden *függetlennek* hívjuk.

A feladatban fellépő, pontokból és az ezeket összekötő élekből álló ábrákat *gráfoknak* nevezzük. A példa alapján világos, hogy egy gráf esetén csak az lényeges, hogy mely pontok vannak éllel összekötve. Így az él alakja, hossza, egyenes vagy görbe volta stb. érdektelen.

A gráfok igen sok helyen előforduló struktúrák. Úthálózatok, vízvezetékek, elektromos hálózatok természetes módon jellemezhetők gráfokkal, de éppen példánkból látható, hogy bármilyen helyzet leírására alkalmasak, ahol bizonyos dolgok és köztük

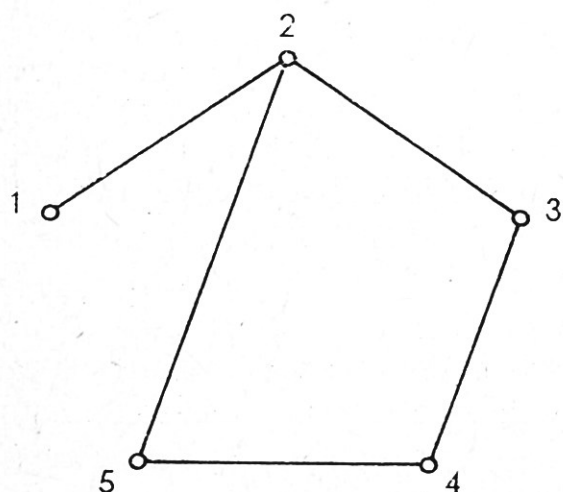
meglevő vagy nem meglevő egyfajta kapcsolat játszik szerepet. A gráfok tulajdonságait a kombinatorika egyik ága, a *gráfelmélet* vizsgálja. Könyvünkben sok példát hozunk fel a gráfokra vonatkozó feladatok köréből, mert sok általános feladattípusnak, mint éppen a most tárgyalt pakolási feladatnak is, a gráfok nyelve szemléletes és egyben jól kezelhető megfogalmazását adja.

Ha a gráfot számítógépben tároljuk, akkor persze nem vihetjük be a rajzát a gépbe. Ehelyett a gráf pontjait az $1, 2, \dots, n$ számokkal megszámozzuk és egy $n \times n$ -es $[a_{ij}]_{i=1}^n_{j=1}^n$ mátrixot képezünk, melyben

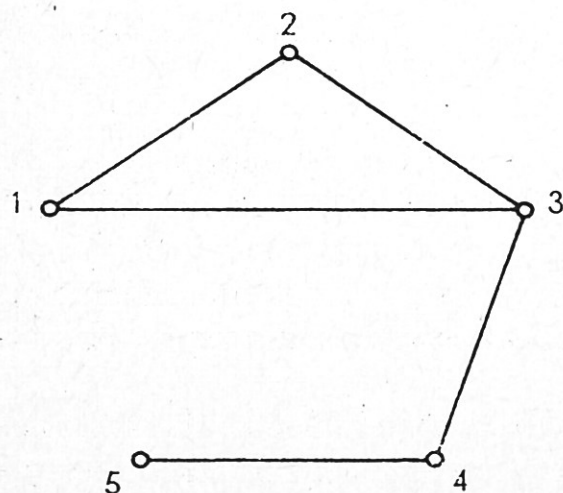
$$a_{ij} = \begin{cases} 1, & \text{ha az } i\text{-edik és } j\text{-edik pont össze van kötve} \\ 0, & \text{ha nem.} \end{cases}$$

Ezt a mátrixot a gráf *adjacencia*- (szomszédossági) mátrixának nevezzük.

A gráf mátrixa ilyenformán a főátlóra nézve szimmetrikus (1. ábra).



1. ábra



2. ábra

Szóba jöhet más tárolási mód is, például megtehetjük, hogy felsoroljuk az összes pontokat és éleket. Pl. az 1, 2, 3, 4, 5 pontokkal és (1, 2), (2, 3), (1, 3), (3, 4), (4, 5) élekkel megadott gráf adjacenciamátrixa és rajza (2. ábra):

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Ha az élek száma a pontszámhoz viszonyítva kicsi, ez utóbbi tárolási mód gazdaságosabb lehet. E könyvben azonban, egyszerűség kedvéért, csak az adjacenciamátrixos tárolással foglalkozunk.

Térjünk vissza tehát a pakolási problémához: *Adott egy G gráf: válasszuk ki maximális számú független pontját!*

1.2.1. Próbálgató módszerek

Az általános pakolási feladatra igazán jó algoritmus nem ismeretes; amit itt adunk, nem tesz mást, mint hogy módszeresen végigvizsgálja az összes független halmazokat.

Legyen x tetszőleges pont. A független halmazok kétfélék: vagy tartalmazzák x -et, vagy nem. Ha S olyan független halmaz, amely x -et tartalmazza, akkor nyilván nem metszi x szomszédainak A halmazát. Így $S - x$ a $G - A - x$ gráfnak egy független pont-halmaza. Ha $x \in S$, akkor S a $G - x$ gráfnak független pont-halmaza.

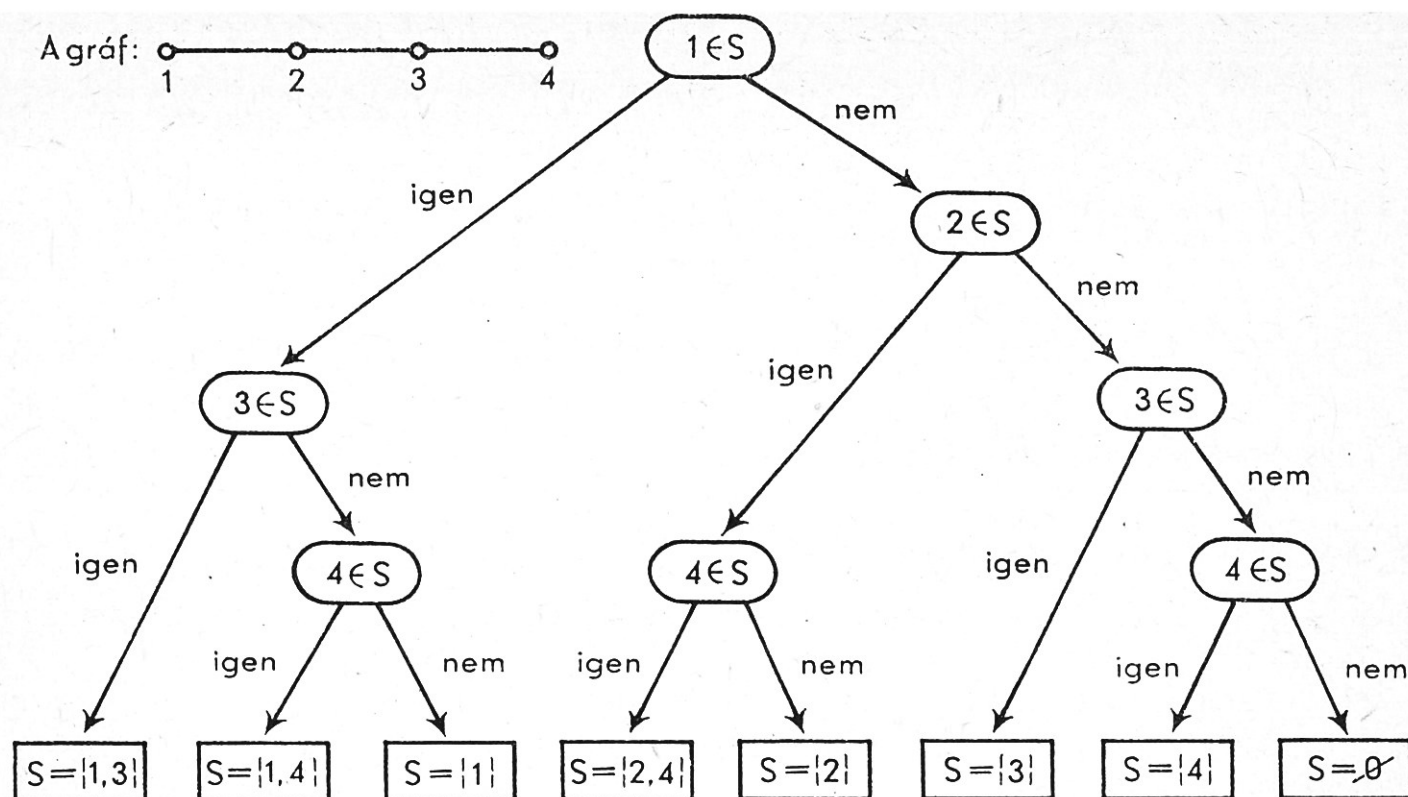
Tehát ha egy maximális elemszámú független pont-halmazt akarunk kiválasztani, a következőképpen járhatunk el.

1. Kiválasztunk egy x pontot, és meghatározzuk szomszédainak A halmazát.

2. Megkeresünk egy-egy maximális elemszámú független halmazt a $G - x$ és $G - A - x$ gráfokban; legyenek ezek S_1 , ill. S_2 . A $G - x$, $G - A - x$ gráfok kisebbek, mint G , így ezekre feltehetjük, hogy már meg tudjuk oldani a pakolási feladatot.

3. Ha S_1 -nek több eleme van, mint S_2 -nek, akkor S_1 , ha ugyanannyi, akkor $S_2 + x$ lesz G -nek maximális független rész-halmaza.

Ez az eljárás igen sok ($\sim 2^n$) lépést igényel, ezért igen jelentősek az olyan fogások, egyszerű észrevételek, melyek ezt rövidítik. Hogy ezeket jobban áttekinthessük, ábrázoljuk az egész eljárást egy ún. *bináris fával*. Algoritmusunk azzal kezdődik, hogy két esetet különböztetünk meg aszerint, hogy x benne van-e a tekintett



3. ábra

halmazban. Ezt a kérdést egy karika ábrázolja felül. Ha pl. igen a válasz, választunk a $G - A - x$ gráfból egy y pontot, és ismét két esetet különböztetünk meg aszerint, hogy y benne van-e a halmazban. A 3. ábra egy 4 pontú gráf példáján mutatja be ezt.

Az algoritmus nem tesz tehát mást, mint felsorolja az összes független halmazokat. Úgy működik, hogy először a tetején kezdve és mindig balra tartva leszaladunk a szögletes blokkig; az S -nek itt megadott értékét tároljuk. Ekkor visszalépünk a legalsó elágazási pontig, és ott jobbra kanyarodunk, majd a továbbiakban ismét balra tartva leszaladunk a szögletes blokkig. Általában minden lépésben visszamegyünk a legalsó olyan elágazásig, melyből még bejáratlan ág indul ki, és azon mindig balra tartva lemegyünk a szögletes blokkig. S -nek ebben megadott értékét összehasonlítjuk az általunk tárolt S halmazzal, és ha az előbbi nagyobb, kicseréljük.

Ez az egyszerű séma érezhetően nem nagyon kötődik a pakolási problémához; bármely esetszétválasztásos eljárás hasonló szerkezetű. Az eljárás neve: *bináris keresés*.

A bináris keresések során gyakran felhasználható fontos elv a következő: ahol nincs, ott ne keress. Más szóval, ha egy ágról valamilyen megfontolásból már a tetején látjuk, hogy a keresett maximális S halmazt nem szolgáltatathatja, akkor nem kell vele foglalkozni. Ilyen eset pl. az, ha az ág semmiképpen sem adhat a már eddig megtalált S halmaznál nagyobb független halmazt. Példánkban a bal szélső $S = \{1, 3\}$ halmaz megtalálása után már

a $3 \in S$ -ből kiinduló, nem-mel megjelölt ágat, valamint a $2 \in S$ -ből kiinduló igen-nel és nem-mel megjelölt ágat nem kell bejárnunk, mert még minden hátralevő ponttal együtt sem kaphatnánk belőlük három független pontot. Az ilyen, nagyságbecsléssel kombinált bináris keresést angolul „branch and bound” módszernek hívják. Ez is igen általános elv, az egyes feladatokból nem sok sajátosságot használ ki.

A pakolási probléma esetében könnyű leszorítani a lépésszámot mintegy $c \cdot 1,381^n$ -re (c egy alkalmas állandót jelent). Lássuk be ezt teljes indukcióval! Az az eset, amikor bármely pont foka, vagyis a hozzá illeszkedő élek száma legfeljebb 2, igen könnyű.

1.9. feladat. Milyen szerkezetűek az ilyen gráfok? Mutassuk meg, hogy maximális független ponthalmazuk $c \cdot n^2$ lépésben megkereshető.

Tegyük fel tehát, hogy a gráfban van egy legalább 3-ad fokú x pont (persze ezt is meg kell keresni, de ez nem igényel több lépést n^2 -nél, ami elhanyagolható az itt rendszerint szükséges lépésszámok mellett). Ekkor az x -et nem tartalmazó független halmazok legfeljebb $c \cdot 1,381^{n-1}$, az x -et tartalmazók $c \cdot 1,381^{n-4}$ lépésben megvizsgálhatók. Ezért nem kell több lépés, mint

$$c \cdot 1,381^{n-1} + c \cdot 1,381^{n-4} = c \cdot 1,381^n \left(\frac{1}{1,381} + \frac{1}{1,381^4} \right) \approx c \cdot 1,381^n.$$

1.10. feladat: Igazoljuk, hogy ha S maximális független ponthalmaz, akkor bármely x pontra vagy x , vagy x -nek valamely szomszédja S -ben van!

1.11. feladat: Bizonyítsuk be, hogy bármely x ponthoz van olyan maximális független ponthalmaz, mely vagy x -et, vagy x -nek legalább két szomszédját tartalmazza!

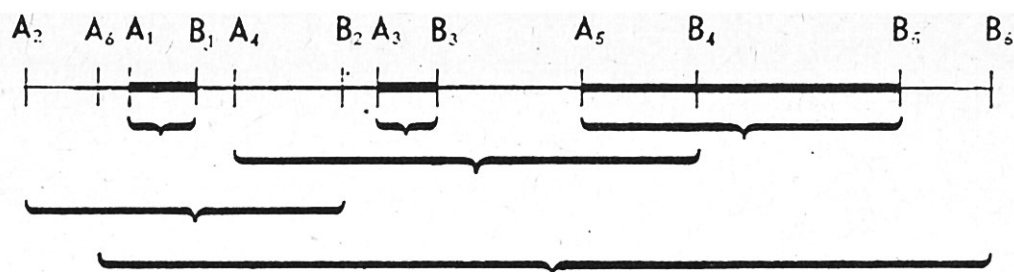
1.12. feladat: Az 1.11. feladat alapján adjunk meg a maximális független halmaz megkeresésére egy legfeljebb $c \cdot 1,3^n$ lépésszámú algoritmust!

1.2.2.

Intervallumpakolás

A pakolási problémának vannak olyan speciális esetei, amik lényegesen gyorsabban megoldhatók, mint az általános feladat. Ilyen eset a 3. példa, a teherautó-szállítások tervezése. A kérdést így modellezhetjük: *adottak egy egyenesen az $A_1B_1, \dots, A_nB_n$ szakaszok* (egyszerűség kedvéért tegyük fel, hogy az A_i, B_j pontok mind különbözőek); *válasszunk ki közülük maximális számú, páronként idegen szakaszt!* Visszatérünk azonban a teherautós megfogalmazásra, mert az szemléletesebb.

Az eljárás, amit ajánlunk, a következő: induljon el a teherautó, és nézze meg, hol kell megrakodnia ahhoz, hogy a lehető leghamarabb megszabaduljon a terhétől. Pl. ha az ábrán látható helyzet alakul ki, akkor az A_1 pontban kell először rakodnia (4. ábra).



4. ábra

Ezt a terhet szállítsa el abba a B_{i_1} pontba, ahova kell, majd mindazon szállítások közül, melyeket még egyáltalán elvégezhet (vagyis melyeknek a kezdőpontját még nem hagyta el), válassza ki azt, amelynek a célja, B_{i_2} , legközelebb van az úton. Hasonlóan, ha már néhány szállítást elvégzett, és éppen a legutóbbi B_{i_r} céljánál lerakodott, akkor tekintse át a még egyáltalán szóba jöhetőket, és azt végezze el, amelyiknek a célja, $B_{i_{r+1}}$ a legközelebb van. Az ábrán vastagon kihúzott szakaszok jelölik a ténylegesen elvégzett szállításokat. Végül a B_{i_k} célpont után már nincs elvégezhető szállítás.

Ez nagyon egyszerű eljárás, de vajon a legjobb eredményt, vagyis a legtöbb diszjunkt szakaszt adja-e? Megmutatjuk, hogy igen. Ehhez azt jegyezzük meg, hogy a $B_{i_1}, B_{i_2}, \dots, B_{i_k}$ pontok minden $A_j B_j$ intervallumot „leszúrnak”, vagyis mindegyik $A_j B_j$ intervallum tartalmazza valamelyiket. Tegyük fel, hogy nem lenne így, hanem az $A_j B_j$ intervallum teljes egészében beleesne pl. a $B_{i_r} B_{i_{r+1}}$ közbe (hasonlóan jutunk ellentmondásra, ha B_{i_1} elé vagy B_{i_k} után esne $A_j B_j$). Ekkor a B_{i_r} célpontban az $A_j B_j$ szállítás még elvégezhető lett volna, sőt mivel B_j közelebb van B_{i_r} -höz, mint $B_{i_{r+1}}$, ezt kellett volna elvégezni. Ez az ellentmondás bizonyítja, hogy $A_j B_j$ tartalmazza valamely B_{i_r} -t.

Ekkor viszont diszjunkt intervallumok bármely rendszere legfeljebb k intervallumból állhat, hiszen mindegyik benne szereplő intervallum a $B_{i_1}, \dots, B_{i_k}$ pontok valamelyikét tartalmazza. Tehát a kiválasztott rendszer valóban maximális.

Jegyezzük meg, hogy a fenti algoritmus a következő eredményt is nyújtotta.

- **Tétel:** Az egyenesen adott szakaszok közül kiválasztható diszjunkt szakaszok maximális száma egyenlő a szakaszokat lefogó pontok minimális számával.

Gyakran előfordul, hogy ha gyors eljárást tudunk adni valamilyen szempontból optimális elem vagy halmaz kiválasztására, akkor ez egyben az optimum nagyságára ilyen egyszerű, de nem triviális formulát is szolgáltat. (Az általános esetre vonatkozó algo-

ritmusainkból semmi hasonlót nem kaptunk.) Ugyanakkor ez a diszjunkt intervallumok és leszűrő pontok számai közti összefüggés tette lehetővé, hogy algoritmusunk helyességét igazoljuk. Hasonló jelenséggel később is találkozunk még.

1.13. feladat: A teherautó minimális számú fordulóval szeretné valamennyi szállítási kötelezettségét teljesíteni. Adjunk a fentihez hasonló egyszerű algoritmust útjának megtervezésére. Mi fogja meghatározni a szükséges fordulók minimális számát?

1.2.3. Párosítások

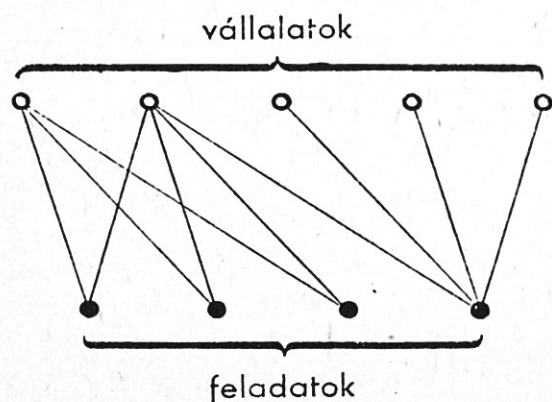
Tegyük fel, hogy n feladatunk van, ezeket m vállalat között kell kiosztanunk. Egy-egy vállalat a feladatok közül csak bizonyosakat (más vállalat másokat) hajlandó elvégezni, és ezek közül is csak egyet. Döntsük el, lehetséges-e a feladatok szétosztása, vagy ha nem, osszunk szét belőlük, amennyit csak tudunk.

A helyzetet egy gráffal írhatjuk le: felül egy-egy ponttal ábrázoljuk a vállalatokat, alul egy-egy ponttal a feladatokat, és minden vállalatot összekötünk mindazokkal a feladatokkal, melyek elvégzésére hajlandó (5. ábra).

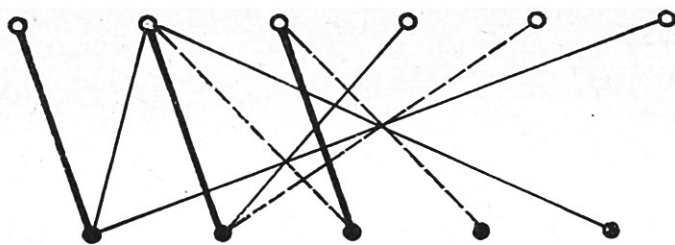
Arra vagyunk kíváncsiak, hogy lehet-e minden feladathoz egy vele összekötött vállalatot rendelni, különböző feladatokhoz különbözőt. Azok az élek tehát, melyek az egyes feladatokat a hozzájuk rendelt vállalatokkal kötik össze, függetlenek, vagyis nincs közös pontjuk. Feladatunkat is megfogalmazhatjuk: döntsük el, van-e annyi független (azaz közös végponttal nem rendelkező) él a gráfban, ahány feladat van, ill. keressük meg a független élek maximális számát.

1.14. feladat: Van-e 4 független él az 5. ábrán?

Az utóbbi megfogalmazásból kitűnik, hogy a pakolási prob-



5. ábra



6. ábra

léma egy speciális esetével állunk szemben: maximálisan sok független élt akarunk „bepakolni” egy páros gráfba, vagyis olyan gráfba, amelynek pontjai két halmazra bonthatók úgy, hogy élek csak a két halmaz között futnak. Ennek elvégzésére azonban az általános algoritmusnál sokkal hatékonyabb (bár az intervallumpakolásnál azért bonyolultabb) algoritmus áll rendelkezésünkre. Ezt „magyar módszernek” is nevezik, mert Kőnig Dénes és Egerváry Jenő eredményeiből volt kiolvasható (ezekre még visszatérünk).

Legyen adva egy páros gráf. Válasszunk ki egymás után független éleket, amíg csak el nem akadunk, vagyis nincs olyan él, amely minden kiválasztott éltől független. Általában nem lesz igaz, hogy máris a lehető legtöbb független élt választottuk ki; pl. a 6. ábrán elég ügyetlenek voltunk: mint látni fogjuk, a megvastagított éleknél többet is kiválaszthattunk volna.

Nevezzük a kiválasztott éleket *fekete éleknek*; mindazon pontokat, amelyeket nem fednek le, *telítetlen pontoknak*.

A gráfban *útnak* nevezünk egy különböző pontokból álló $(x_0, \dots, x_k)$ sorozatot, ha x_i -ből x_{i+1} -be él megy $(i=0, 1, \dots, k-1)$. Egy önmagát nem metsző utat alternáló útnak hívunk, ha egy alsó telítetlen pontból indul, és minden második éle fekete él. A módszer lényege a következő nyilvánvaló észrevétel: *ha egy alternáló út felső telítetlen pontban végződik, akkor a rajta fekvő fekete éleket az út többi élére kicserélve, a független élek számát növeljük* (6. ábra).

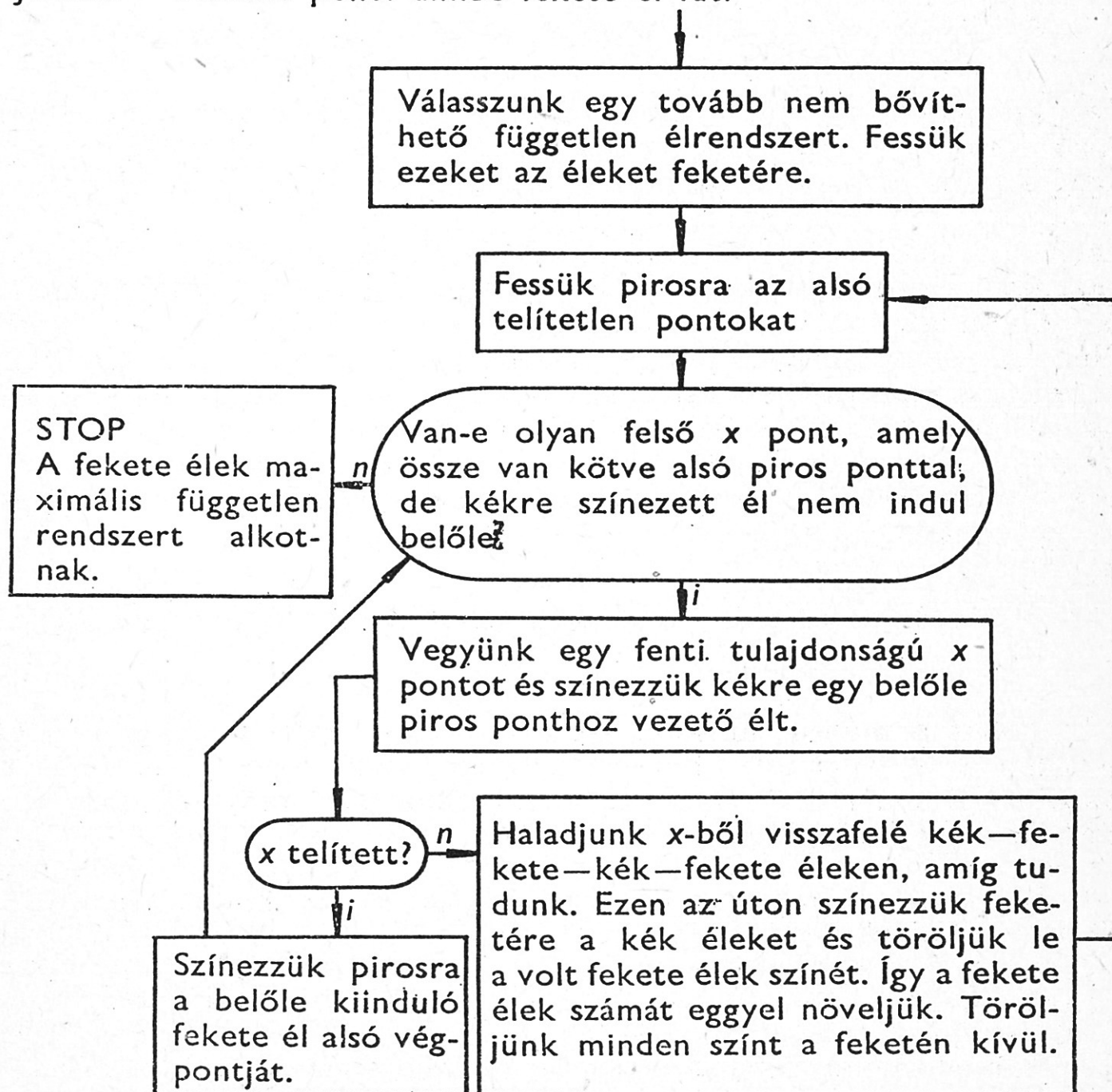
Megmutatjuk, hogy ha ilyen módon nem tudjuk növelni a független élek számát, akkor a fekete élek máris maximális független érendszer kell hogy alkossanak. Jelölje ugyanis X mindazon fenti pontok halmazát, melyek elérhetők alternáló úttal, és tegyük fel, hogy X -ben nincsen telítetlen pont. Jelölje X' az X -ből kiinduló fekete élek alsó végeinek halmazát, és X'' a többi alsó telített pont összességét. Ekkor $X \cup X''$ minden élt leszúr. Legyen ugyanis (a, b) egy él és a annak alsó végpontja. Ha az a telítetlen pont, akkor b definíciószerűen X -be tartozik. Ha a az X' -ben van, akkor van olyan P alternáló út, mely a -ban végződik. De ekkor olyan is van, amely b -ben végződik: ugyanis b vagy rajta van P -n, akkor P -nek b -ig terjedő szakasza, vagy nincs rajta, és akkor $P + (a, b)$ ilyen alternáló út. Tehát ekkor b az X -ben van. Végül ha a az X'' -ben van, nincs is mit bizonyítani.

Ha $X \cup X''$ minden élt leszúr, akkor nem is lehet több független élt megadni, mint ahány eleme $X \cup X''$ -nek van. De ez éppen a fekete élek száma; a fekete élek tehát máris a lehető legtöbbször vannak.

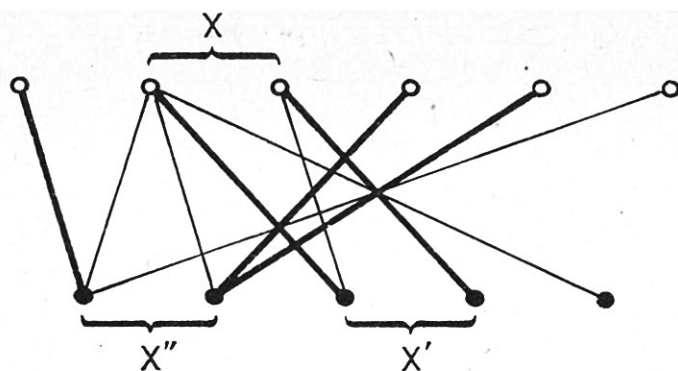
Bemenet: G páros gráf.

Kimenet: G egy maximális független érendszer.

Jelölés: Telített pont: amibe fekete él fut.



Most már csak arra kellene eljárást adni, hogy eldöntsük, van-e telítetlen felső pontban végződő alternáló út. Először is színezzük pirosra a telítetlen alsó pontokat. Ezután minden lépés-



7. ábra

ben keresünk egy olyan felső x pontot, mely össze van kötve piros alsó ponttal, de még kékre színezett él nem indul ki belőle, és egy belőle piros alsó ponthoz menő élt kékre színezünk. Ha x telített, akkor a belőle kiinduló fekete él alsó végpontját pirosra színezzük, és az eljárást ismételjük. Ha x telítetlen, akkor megállunk. Ebben az esetben x -ből kék—fekete—kék—fekete... úton visszafelé haladva egy olyan alternáló utat találunk, amely x -ben végződik, és a fekete élek számát növelni tudjuk. A másik eset az, amikor azért akadunk el, mert x pontot nem találunk. Könnyen igazolható, hogy ebben az esetben telítetlen felső pontban végződő alternáló út nincs, tehát a fekete élek máris maximális független élrendszert alkotnak.

A feladatban szereplő páros gráfot gyakran egy $[a_{ij}]_{i=1}^n_{j=1}^m$ mátrix írja le, például úgy, hogy „lent” n pont, „fent” m pont van és

$$a_{ij} = \begin{cases} 1, & \text{ha a lenti } i\text{-edik pont szomszédos a felső } j\text{-edikkel;} \\ 0, & \text{ha nem.} \end{cases}$$

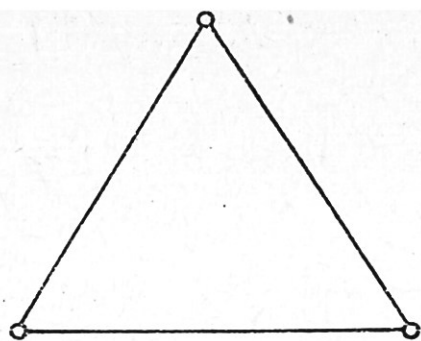
Tehát a mátrix minden 1-ese egy élnek felel meg. Ebben az esetben arra vagyunk kíváncsiak, hogy hány olyan 1-es választható ki a mátrixból, melyek közül semelyik kettő sincsen egy sorban vagy oszlopban.

1.15. feladat: Fogalmazzuk át a párosítási problémára megadott algoritmust mátrixokra.

Érdemes a fenti algoritmusból kiolvasni az alábbi tételt:

- **Kőnig tétele:** *Bármely páros gráf független éleinek maximális száma egyenlő az éleket leszűrő pontok minimális számával.*

Már megjegyeztük, hogy egy ilyen, a keresett optimumot megadó formula a gyors megoldó algoritmus létezésével általában együtt jár, bár a két tulajdonság ekvivalens volta nincs bizonyítva.



8. ábra

Egy másik, fontos tétel, mely algoritmusunkból kiolvasható:

- **Hall tétele:** *Egy páros gráfban akkor és csak akkor lehet minden „alsó” ponthoz egy vele összekötött „felső” pontot rendelni (különböző „alsó” pontokhoz különbözőt), ha bármely k -ra bármely k darab alsó pont legalább k felső ponttal van összekötve.*

Felvetődik a kérdés: hogyan kell egy tetszőleges (tehát nem páros) gráfban maximálisan sok független élt keresni. Itt a helyzet bonyolultabb, amit az is mutat, hogy *Kőnig* tétele ilyen formában nem marad igaz. Pl. a 8. ábrán látható „háromszög” gráfban nincs két független él, de az összes élék leszúrásához legalább 2 pont kell. *Kőnig* tételének megfelelő általánosítása *Tutte* és *Berge* nevéhez fűződik; hatékony algoritmust (mely n pontú gráf esetén kb. n^3 lépésben ér célhoz) *Edmonds* adott. Ezt az algoritmust magyar nyelven is megtalálhatja az olvasó *Busacker—Saaty* idézett könyvében.

1.3.

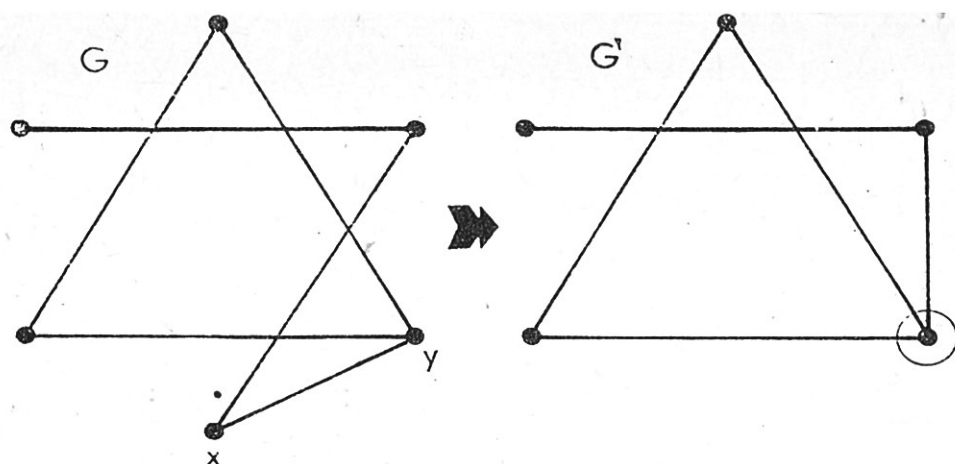
Útkeresés

1.3.1.

Összefüggőség

Lehet-e Budapestről a világ bármely fővárosába telefonálni? Hogy erre választ adhassunk, tudnunk kell, hogy mely fővárosok között van közvetlen telefonvonal. Így ismét egy gráffal leírható helyzet áll elő: pontjaink a világ összes fővárosai és minden él egy közvetlen telefonvonalat jelöl. Kérdés: össze lehet-e kötni bármely két fővárost közvetlen telefonvonalakból álló „lánccal”?

Fogalmazzuk meg a feladatot pontosan a gráfok nyelvén! A G gráfot *összefüggőnek* nevezzük, ha bármely pontból bármely pontba eljuthatunk úton. Példánkban alapvető kérdés, hogy a gráf összefüggő-e.



9. ábra

Egy egyszerű eljárásra ad lehetőséget az alábbi észrevétel: Tekintsük a gráf egy élét; legyenek ennek végpontjai x és y . Ezt az élt röviden (x, y) élnek hívjuk. Jelöljük G' -vel az (x, y) él összehúzásával kapott gráfot: ebben x -et és y -t egy új ponttal helyettesítjük, és ezt az új pontot mindazon régi pontokkal összekötjük, melyekkel akár x , akár y össze volt kötve (9. ábra). Megfigyelhetjük, hogy G' akkor és csak akkor összefüggő, ha G az.

Ha ezt az ötletet lefordítjuk az adjacenciamátrix nyelvére, az alábbi eljárást olvashatjuk ki belőle. Jelölje n a pontok számát. Tegyük fel, hogy az első sorban van egy pozitív elem, mondjuk az i -edik helyen. Adjuk hozzá az első sort az i -edikhez, majd az első oszlopot az i -edikhez, „Boole-féle értelemben”, vagyis a

$$0 + 0 = 0, \quad 0 + 1 = 1 + 0 = 1 + 1 = 1$$

szabályok szerint. Hagyjuk el az első sort és oszlopot, és az i -edik sor i -edik helyén létrejött 1-est változtassuk 0-ra. Ismételjük meg az eljárást. Ha közben eljutunk egy 1×1 -es mátrixhoz, a gráf összefüggő. Ha valahol elakadunk, mert az első sorban csak 0-k állnak, a gráf nem összefüggő.

Az itt ismertetett algoritmus lépésszáma n^2 -tel arányos (az arányossági tényező attól függ, hogy mit tekintünk egy lépésnek). Megmutatjuk, hogy ennél lényegesen gyorsabb eljárás nem is adható. Tegyük fel ugyanis, hogy gráfunk egyenként két, $\frac{n}{2}$ elemű részre bontható, melyek között él nem halad, de az egyes részekben első pontok össze vannak kötve. Ekkor a gráf nem összefüggő. Képzeld el, hogy algoritmusunk lefutott és kimutatta, hogy a gráf nem összefüggő. Ekkor az algoritmusnak meg kellett vizsgálnia mind az $\frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$ hiányzó élt, hogy nincs-e behúzva; ha bármelyikről is nem volna információnk, akkor ez az él akár

be is lehetne húzva és mi nem tudhatnánk biztosan, hogy a gráf nem összefüggő. Így a szükséges lépésszám legalább $\frac{n^2}{4}$. (A kérdésre később még visszatérünk.)

1.3.2.

A legrövidebb út

A telefonhálózat összefüggőségének kérdésénél talán még fontosabb a legrövidebb összeköttetésé. Ez gráfokra így hangzik: Adott egy gráf, és benne két pont, a és b . Keressük meg az a -ból b -be vezető legrövidebb utat!

Érdekes módon a feladat könnyebbé válik, ha többet kíván: Minden x ponthoz keressünk egy a -ból x -be vezető legrövidebb utat! Ezt úgy tesszük, hogy az $i=1, 2, \dots$ értékekre rendre megkeressük azokat a pontokat, melyek i hosszúságú úton elérhetők.

Az a -ból 1 hosszúságú úton (élen) elérhető pontokat persze könnyen megtaláljuk. Jelöljük meg ezeket az 1-es számmal és színezzük az a -ból hozzájuk vezető éleket pirosra.

Keressük meg most mindazokat a pontokat, melyeknek még nincs számuk, és melyek 1-es pontból élen elérhetők. Minden ilyen pontot jelöljük meg 2-essel, és színezzünk pirosra egy-egy beléjük befutó élt. Mivel egy a -ból kettő hosszúságú úton elérhető pont az 1-es jelű pontok valamelyikén át érhető csak el, így 2-essel pontosan azokat a pontokat jelöltük meg, melyek a -ból elérhetők 2, de nem 1 hosszúságú úton.

Hasonlóan folytatva, ha már az 1, 2, $\dots$, i számokkal megcímkeztük a megfelelő pontokat, akkor jelöljük $(i+1)$ -gyel azokat a pontokat, amelyeknek még nincsen számuk, és amelyek elérhetők egy i -vel jelzett pontból élen; majd fessünk pirosra minden ilyen ponthoz egy-egy olyan élt, mely i -vel jelzett pontból fut bele.

Az eljárás akkor ér véget, amikor nincsen $(i+1)$ -gyel megjelölendő pont. Vegyük a b pontot; ha ez nem kapott számot, akkor nem érhető el a -ból úton. Ha kapott, akkor ez az i szám lesz az a -ból b -be vezető legrövidebb út hossza, és egy ilyen utat kapunk, ha a piros éleken át b -ből rendre $i-1, i-2, \dots$ jelzésű pontokon keresztül visszamegyünk a -ba.

Eljárásunkat könnyen átvihetnénk arra az esetre, amikor az egyes élek különböző hosszúságúak, és nem az út éleinek számát, hanem éleinek összhosszát akarjuk minimalizálni. Megengedhetnénk azt is, hogy minden élnek egy kitüntetett iránya legyen;

ilyenkor a keresett útnak minden élen az adott irányban kell végighaladnia.

A legrövidebb utat kereső eljárás sokkal több esetben alkalmazható, mint ahogyan azt első látásra gondolnánk. Ezt az „autóverseny” játékon mutatjuk be. A játékot kockás papíron játsszák. (Mi a négyzetháló metszéspontjairól, mint egy koordináta-rendszer egész koordinátájú pontjairól — „rácspontjairól” — fogunk beszélni. Erre berajzolják egy kanyargós „autóút” két szélét, valamint a rajt- és célvonalat. A versenyző rácsponttól rácspontra lép, nem feltétlenül szomszédosra, de a pályáról nem szabad lelépnie. Minden lépéséhez tartozik egy vektor — a pillanatnyi sebességvektor —, amely a lépés kiindulópontjából az érkezési pontba mutat. A játékszabály az, hogy a lépéshez tartozó sebességvektor az előző lépés sebességvektorától függőleges és vízszintes vetületében is legfeljebb 1 egységnyit különbözzék. Ezt úgy a legegyszerűbb betartani, ha — mint a 10. ábra kinagyított útrészlete mutatja — az előző lépés vektorát az új kezdőpontból indítva pontozva megismételjük. Ekkor az új lépés vektorának végpontja a pontozott vektor végétől mindkét koordinátában csak 1 egységnyit térhet el. Tehát az autó gyorsulása (de lassuló- és forduló-képessége is; ettől ravasz a játék!) korlátozott. A cél természetesen: minél rövidebb idő alatt érni el a célvonalat.

A versenyző útja nem gráfút, hiszen az, hogy egy lépése jogos-e, attól is függ, mit lépett előzőleg. Tudjuk azonban a fizikából, hogy egy mozgó tömeget *helyének és sebességének* megadásával lehet teljesen leírni. Ez azt sugallja, hogy tekintsük a (metszéspont, sebesség) párokat vagy, ami ezzel egyenértékű, az olyan (a, b) párokat, melyek mindkét eleme metszéspont; ezek azt fejezik ki, hogy „a versenyző a b pontban van, ahova az a pontból lépett”. Felrajzolunk tehát egy gráfot, melynek pontjai az összes (a, b) pontpárok, ahol a és b az autóút rácspontjai. Ha a versenyző ebben a gráfban az (a, b) pontban tartózkodik, akkor azon (b, c) pontok bármelyikére léphet, melyekre a $\vec{bc}$ „pillanatnyi sebességvektor” az előző $\vec{ab}$ „pillanatnyi sebességvektortól” bármely koordinátájában legfeljebb 1-gyel különbözik. Így tehát, ha minden (a, b) pontot irányított éllel összekötünk, mindazon (b, c) pontokkal, amelyekre $(-1, -1) \leq \vec{ab} - \vec{bc} \leq (1, 1)$, az eredményül kapott irányított gráfban a versenyző lehetséges mozgásai éppen azok az utak, melyek az (a_0, a_0) pontból indulnak ki (ahol a_0 a versenyző indulási pontja). Ezek közül a legrövidebb olyan, mely „áthalad a célon”, könnyen meghatározható a fentebb leírt algoritmussal.

1.3.3.

A leghosszabb út

Keressük meg a gráf két pontja közti leghosszabb utat! Ez a probléma kevésbé tűnik természetesnek, mint az előző. De gondoljunk csak arra, ha azt kérdezzük: „Be lehet-e járni a gráfot (adott pontból indulva és adott pontban végezve) úgy, hogy minden pontot egyetlenegyszer érintünk?” — ez a kérdés az előbbinek speciális esete.

Legyen tehát a, b két adott pont a G gráfban. Felvetődik a kérdés, hogy nem lehet-e a leghosszabb (a, b) utat a legrövidebb (a, b) út mintájára megkeresni? Az a pont szomszédaihoz mindenestre vezet 1 hosszúságú út; másodsomszédaihoz 2 hosszúságú stb. De vezethet hosszabb is! Hiába találtunk már a -ból pl. b -be vezető utat, azt megállapítani, hogy ez leghosszabb-e, közvetlenül nem tudjuk. Itt nem is ismert — és a 2. fejezetben ismertetésre kerülő elmélet azt sugallja, hogy nem is adható — olyan rövid algoritmus, mint a legrövidebb út problémájára; valószínű, hogy minden algoritmus — a legkedvezőtlenebb esetben — exponenciálisan hosszú időt igényel.

Kézenfekvő algoritmus a következő: A b pont szomszédai legyenek $b_1, \dots, b_k$. Jelölje továbbá $D_G(x, y)$ a G gráfban az x, y pontokat összekötő leghosszabb út hosszát. Mivel a b pontot csak a $b_1, \dots, b_k$ pontok valamelyikén keresztül érhetjük el, és egy (a, b) út a és b_i közötti szakasza nem haladhat át b -n, nyilván fennáll, hogy

$$D_G(a, b) = 1 + \max_{1 \leq i \leq k} D_{G-b}(a, b_i). \quad (1)$$

A $D_{G-b}(a, b_i)$ számokat hasonlóan kifejezhetjük az a -t a b_i -k szomszédaival összekötő leghosszabb út hosszával (egy még egy ponttal kisebb gráfban) stb. Ez az egyszerű eljárás azonban nem más, mint a b -ből kiinduló összes lehetséges utak számbavétele. Ha G pl. teljes n -gráf, vagyis minden pont minden ponttal össze van kötve, akkor a lépésszám kb. $(n-2)!$, ami igen nagy. Ha G fokszámai kicsinyek, akkor az eljárás jobban működik, de semmiképpen nem valami jó.

Nem is várhatunk többet olyan eljárástól, amely ennyire nem vesz tekintetbe semmilyen speciális észrevételt vagy tulajdonságot. De hát úgy tűnik, ezeket nem is lehetne hatékonyan figyelembe venni.

Ilyen nagy lépésszám esetén különösen nagy jelentősége van az olyan elemi fogásoknak, egyszerűsítő észrevételeknek, melyekre a legrövidebb út keresésekor nem fordítottunk figyelmet. Noha az algoritmus „lényegét” a fenti (1) képlettel kifejeztük,

érdemes egy kicsit részletesebben leírni ugyanazt. A szemléletesség kedvéért úgy beszélünk, mintha a gráf rajzán dolgoznánk színes ceruzával, de persze tudnunk kell, hogy ez már pl. 10 darab 3-adfokú pont esetén is kézzel elvégezhetetlenül hosszú munka volna.

Érdemes először megnézni, hogy van-e egyáltalán (a, b) út, és ha van, rögzítsünk egyet, Q -t.

Rajzunk minden esetben a következő: egy b -ből kiinduló P út élei pirosak, ezenfelül néhány, erről elágazó él kék (ez azt akarja jelenteni, hogy arra már próbáltunk menni). Jelölje P végpontját x .

1. Kezdetben $x \leftarrow b$.

2. Ha $x \neq a$ és van x -nek olyan y szomszédja, mely nincs P -n és melyre az (x, y) él szintelen, színezzük ezt pirosra és legyen $x \leftarrow y$.

3. Ha $x \neq a, b$ és x minden szomszédja vagy P -n van, vagy kék él köti össze x -szel, akkor radírozzuk le az x -ből kiinduló élekről a színezést és lépjen x visszafelé egyet P -n. A P út utolsó élet fessük kékre.

4. Ha $x = a$, akkor hasonlítsuk össze P -t Q -val, és ha P hosszabb, tegyük meg P -t új Q -nak, és lépünk vissza 1-re. Ha P nem hosszabb, mint Q , járjunk el úgy, mint 3-ban.

5. Ha $x = b$ és minden x -ből kiinduló él már kék, álljunk le. Az éppen feljegyzett Q út lesz a leghosszabb.

Nézzünk most néhány olyan fogást, ami itt alkalmanként gyorsíthatja az eljárást. Könnyen meg tudjuk határozni minden lépésben azokat a pontokat, melyek a -ból elérhetők P érintése nélkül. Nyilvánvaló, hogy P -t csak ilyen pont irányába érdemes növelni. Tehát ha x -ből, azon pontok halmazához, melyek a -ból P érintése nélkül elérhetők, csak kék él vezet, éppúgy visszalépünk, mint a korábbi 3. lépésben. Ez külön fáradsággal jár, de sok bolyongást megtakarítunk vele olyan részeken, ahonnan már úgyse juthatnánk el a -ba. További előnye, hogy a P út mindig kiterjeszthető lesz (a, b) úttá.

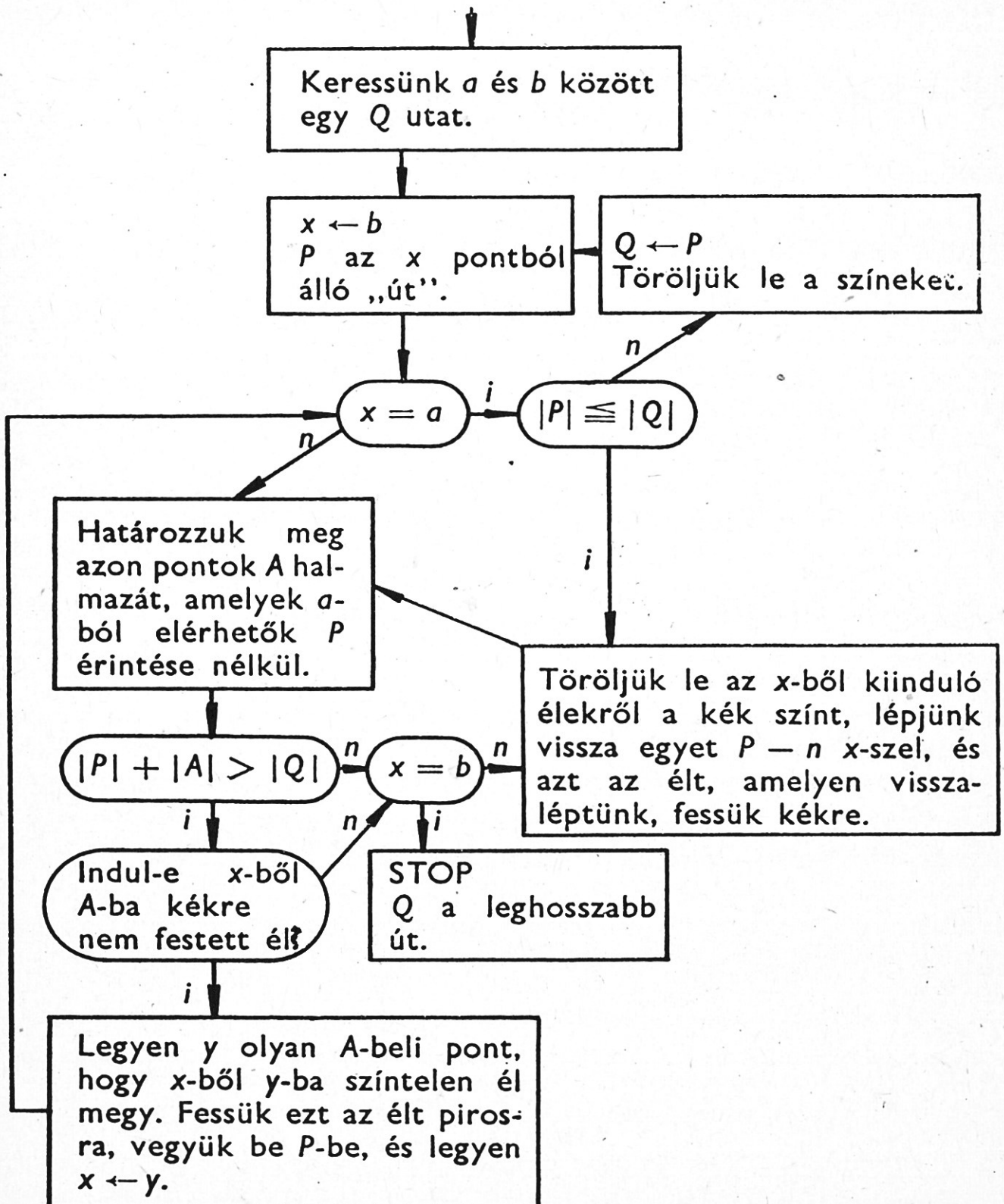
A P úton levő és az a -ból P érintése nélkül elérhető pontok együttes száma felső becslést ad a P -t folytató leghosszabb út hosszára. Ha Q ennél hosszabb, nem érdemes P -t folytatni, és ismét visszaléphetünk egyet x -szel.

Valójában itt is olyan „branch and bound” típusú algoritusról van szó, mint a pakolási probléma esetében. Mint ott már jeleztük, az ilyen algoritmus nem használ fel lényeges strukturális tulajdonságot, csak az (elvileg nyilvánvaló) esetszétválasztások ügyes és gondos ütemezését valósítja meg.

Jelölés: $|P|$ a P út, $|Q|$ a Q út hossza. $|A|$ az A halmaz elemeinek száma.

Bemenet: G gráf és benne a, b pontok.

Kimenet: Q lesz a leghosszabb út a és b között.



1.4.

Szuboptimális algoritmusok

Az algoritmusok által megoldandó feladatok tekintélyes része egy valamilyen szempontból optimális dolog kiválasztását kívánja. Például a pakolási problémánál maximalizálni kellett a kiválasztott pontok számát azon feltétel mellett, hogy közülük semelyik kettő se legyen összekötve. Az optimumot szolgáltató eljárás lehet igen hosszú, s így lefuttatása igen költséges vagy éppenséggel lehetetlen. Előfordulhat, hogy nincs is igazán szükségünk a pontos optimumra; elegendő olyan „dolgot” találni, amely nincsen messze a pontos optimumtól. Ez gyakran már egészen egyszerű, kézenfekvő módszerrel is elérhető.

Kezdjük egy egészen egyszerű példával, a párosítási problémával. Bár erre viszonylag gyors eljárás adható, nézzünk egy egészen triviálisat: válasszunk ki független éleket, amíg csak tudunk.

Tegyük fel, hogy az $(x_1, y_1), \dots, (x_k, y_k)$ éleket sikerült kiválasztani, amikor elakadtunk. Az persze nem biztos, hogy ez a lehető legtöbb független él. Például ha egy három élből álló út középső élet választjuk ki elsőnek, máris elakadunk, pedig máshogy kezdve két élt is kiválaszthattunk volna. Észrevehetjük azonban, hogy az $x_1, \dots, x_k, y_1, \dots, y_k$ pontok minden élt leszúrnak (ezért akadunk el), így $2k$ ponttal az összes él leszúrható. Ezért $2k$ -nál több független él nem lehet.

Az eljárás tehát a független élek maximális számának legalább felét kiadja.

1.4.1.

A hátizsákfeladat

Tekintsük a következő feladatot: Van n tárgyunk, melyek térfogata rendre $V_1, V_2, \dots, V_n$. Ezeket hátizsákokba akarjuk elhelyezni. Egy-egy hátizsák térfogata V , és semmi más nem korlátozza a bepakolást, mint hogy egy-egy hátizsákba legfeljebb V össztérfogatú dolog kerülhet. Célunk az, hogy minél kevesebb hátizsákba csomagoljuk be a dolgokat.

A gyakorlatban bizonyos időbeosztási kérdések gyakran vezetnek ehhez a problémához. Pl. egy szerkesztőségbe n darab gépelnivaló kézirat érkezik, melyek oldalszáma rendre $V_1, \dots, V_n$. Hány gépírónt kell beállítani, ha azt akarjuk, hogy a munkát 1 nap alatt elvégezzék, tudva azt, hogy egy gépírónt 1 nap alatt csak V oldalt tud gépelni?

Visszatérve ismét a hátizsákos megfogalmazáshoz, nyilvánvaló, hogy a berakodáshoz legalább $(V_1 + \dots + V_n)/V$ hátizsák kell; nem biztos azonban, hogy dolgokat úgy tudjuk illesztgetni, hogy ennyi elég is. Az optimális pakolás meghatározása általában nem könnyű feladat. Sok eljárás ismeretes, de a legrosszabb esetben mindegyik igen sokáig (exponenciálisan hosszú ideig) dolgozik. Ezekre most nem térünk ki, hanem helyette egy igen egyszerű „szuboptimális” elhelyezést adó algoritmust elemzünk. Az eljárást, amit tárgyalunk, „sietős” algoritmusnak fogjuk nevezni. A hátizsákokat sorbarakjuk: egyenként vesszük kézbe a $V_1, V_2, \dots$ térfogatú tárgyakat és mindegyiket beletesszük az első olyan hátizsákba, amelyikbe belefér.

Így általában nem a leggazdaságosabb csomagolást találjuk meg. Legyenek pl. a tárgyak térfogatai

$$\begin{array}{ccc} 1 - 2\varepsilon, \dots, 1 - 2\varepsilon, & 2 + \varepsilon, \dots, 2 + \varepsilon, & 3 + \varepsilon, \dots, 3 + \varepsilon, \\ 6k \text{ db} & 6k \text{ db} & 6k \text{ db} \end{array}$$

ahol ε kicsi pozitív szám (pl. $\varepsilon = 1/2k$) és legyen a hátizsákok térfogata 6 egység. Ekkor minden fajtából 1—1 tárgyat véve, az pontosan belefér egy hátizsákba, így az összes tárgy belefér $6k$ darab hátizsákba. Másrészt a „sietős” eljárással először k számú hátizsákba teszünk 6—6 darabot az első fajta tárgyból; majd $3k$ további hátizsákba 2—2 darabot a másodikból; végül $6k$ hátizsákba 1—1 darabot a harmadikból. Így $6k$ helyett $10k$ hátizsákot használunk.

Bebizonyítjuk azonban, hogy a „sietős” eljárás nem igényel több hátizsákot, mint a feltétlenül szükségesnek a kétszerese. Jelölje m azon hátizsákok számát, amelyekbe a sietős eljárással kerül valami.

- *Állítás:* $m \leq 2(V_1 + \dots + V_n)/V$.

Azt látjuk be ennek igazolására, hogy bármely két hátizsák, melybe egyáltalán került valami, együttvéve több, mint V térfogatú dolgot tartalmaz. Legyenek ezek ugyanis az i -edik és a j -edik hátizsákok ($i < j$), és legyen bennük W_i , ill. W_j térfogatú dolog a csomagolás végén. Tekintsük a j -edik hátizsákba utoljára tett tárgyat. Ennek térfogata legyen V_k . Ekkor nyilván

$$V_k \leq W_j.$$

Másrészt V_k -t nem tudtuk már az i -edik hátizsákba tenni; mivel az i -edik hátizsákban még legalább $V - W_i$ hely volt,

$$V_k > V - W_i,$$

és így

$$W_j \geq V_k > V - W_i, \quad W_i + W_j > V.$$

Legyen most az i -edik hátizsák az, amely a legkevesbé van tele. Ha ez több mint félig van, vagyis $W_i \geq V/2$, akkor minden j -re

$$W_j \geq W_i > V/2,$$

és így

$$V_1 + V_2 + \dots + V_n = W_1 + W_2 + \dots + W_m > m \frac{V}{2},$$

vagyis

$$m < 2(V_1 + \dots + V_n)/V.$$

Ha $W_i \leq V/2$, akkor az előbb bizonyítottak szerint minden $j \neq i$ -re

$$W_j > V - W_i.$$

Tehát

$$\begin{aligned} V_1 + \dots + V_n &= W_1 + \dots + W_m > W_i + (m-1)(V - W_i) = \\ &= (m-1)V - (m-2)W_i \geq (m-1)V - (m-2)\frac{V}{2} = m\frac{V}{2}, \end{aligned}$$

amiből ugyanaz adódik, mint fentebb.

Az állításból az is látható, hogy ha a pontos megoldás, vagyis az adott dolgok bepakolásához szükséges hátizsákok minimális száma m_0 , akkor $m/m_0 < 2$.

Megjegyezzük, hogy *Demers, Garey, Johnson, Graham és Ullman* igazolták azt is, hogy $m/m_0 \leq 1,7$ és példát mutattak arra is, hogy az 1,7 tovább már nem javítható. Ugyanők megvizsgálták a „sietős” csomagolási algoritmus következő változatát; rendezzük a tárgyakat térfogat szerint csökkenő sorrendbe, vagyis legyen $V_1 \geq V_2 \geq \dots \geq V_n$ és ezután alkalmazzuk a „sietős” eljárást. Azt kapták, hogy az algoritmus által felhasznált hátizsákok m' számára $m'/m_0 \leq 1,22\dots$ teljesül, tehát az egyszerű előkészítés, hogy csökkenőleg rendezzük a tárgyakat, lényeges javulást eredményezett, legalábbis a „legrosszabb esetet” illetően. Az átlagos viselkedésről a számítógépes tapasztalatok hasonló képet mutatnak, de erre vonatkozóan elméleti eredmények még nincsenek.

1.4.2.

A fedési feladat

A fedési feladatot két példán mutatjuk be.

1. példa: Legyenek $F_1, \dots, F_m$ az elvégzendő feladatok; $V_1, \dots, V_n$ vállalatok, melyek mindegyike e feladatok közül

néhányat el tud végezni. Ki kell választani minimális számú vállalatot úgy, hogy ezek együttvéve valamennyi feladat elvégzésére alkalmasak legyenek. (Figyeljünk föl a különbségre a párosítási problémához képest; itt egy vállalat több feladatot is kaphat egyidejűleg!)

2. példa: Egy társaságban mindenki hajlandó pingpongozni valakivel, de nem szükségképpen mindenkivel (a hajlandóság kölcsönös). Hogyan kell minimális számú játszmát lejátszani úgy, hogy mindenki legalább egyet játsszon?

E feladatok közös megfogalmazása a következő: Adott egy S halmaz; ennek elemeit pontoknak nevezzük. Adottak továbbá az $E_1, \dots, E_n$ halmazok, melyek egyesítése tartalmazza S -t. Válasszunk ki az $E_1, \dots, E_n$ halmazok közül minimális számút úgy, hogy ezek együtt lefedjék S -t. Ez a fedési probléma. Az első példában S a feladatok halmaza, E_i pedig a V_i vállalat által elvégezhető feladatoké; a 2. példában S a társaság tagjainak összessége, $E_1, \dots, E_n$ pedig azok a párok, melyek hajlandók egymással játszani.

A fedési probléma megoldására sok algoritmus ismeretes, amelyek azonban mind exponenciálisan hosszú ideig működnek. (Ezeket itt nem ismertetjük; egy részük a pakolási problémánál tárgyalt „branch and bound” módszerrel dolgozik, más részük pedig a lineáris programozás módszerére támaszkodik, amelyeket e könyvben nem tudunk tárgyalni.) Helyettük egyszerű szuboptimális algoritmusra mutatunk példát. Ez olyankor működik jól, ha az E_i halmazok viszonylag kicsik. Példaként itt arra az esetre szorítkozunk, amikor minden E_i 3 elemű.

Miért nem a 2 elemű E_i -k esetét tekintjük? (Ez szerepel a 2. példában is!) A válasz: a 2 elemű E_i -k esete visszavezethető a párosítási feladatra. A következő eljárás adható ugyanis: Válasszunk ki az adott E_i (kételemű) halmazok közül maximálisan sok diszjunktat; legyenek ezek $E_{i_1}, \dots, E_{i_k}$. (Ez éppen a párosítási feladat, és mint ott megmutattuk, eléggé hatékonyan megoldható.) Ha vannak még olyan pontok, melyeket $E_{i_1}, \dots, E_{i_k}$ nem fednek le, válasszunk ki minden ilyen ponthoz egy-egy őt tartalmazó E_j halmazt. Ha S elemszáma m , akkor így $m - 2k$ további halmazt kell kiválasztanunk, vagyis az összes kiválasztott halmazok száma $k + (m - 2k) = m - k$. Ezen a módon az S -et lefedő lehető legkevesebb halmazt választottuk ki.

1.16. feladat: Igazoljuk ezt az állítást!

Így tehát kézenfekvő, hogy arra az esetre keresünk szuboptimális algoritmust, amikor minden egyes E_i 3 elemű.

Válasszunk ki egy E_{i_1} -et; majd, ha van ilyen, egy E_{i_2} -t, mely E_{i_1} -től diszjunkt; folytassuk ezt, mindig olyan $E_{i_{k+1}}$ -et választva, melynek a már kiválasztott $E_{i_1}, \dots, E_{i_k}$ halmazokkal nincs közös pontja.

Ha elakadtunk, akkor nézzük meg, van-e olyan az adott E_i halmazok között, mely 2, még lefedetlen pontot is tartalmaz, és ha igen, tegyük bele ezt a rendszerbe. Folytassuk ezt, míg el nem akadunk, vagyis bármely E_j a még be nem fedett pontok közül már csak egyet tartalmaz.

Ekkor minden, még lefedetlen ponthoz válasszunk ki egy-egy őt tartalmazó E_j halmazt. Az így kiválasztott halmazok tehát minden pontot le fognak fedni.

Röviden szólva, az algoritmus minden lépésben olyan E_i halmazt választ, amely a még lefedetlen pontokból a lehető legtöbbet fedi le. Ez magyarázza, hogy ezt az algoritmust „mohó” algoritmusnak (greedy algorithm) nevezzük.

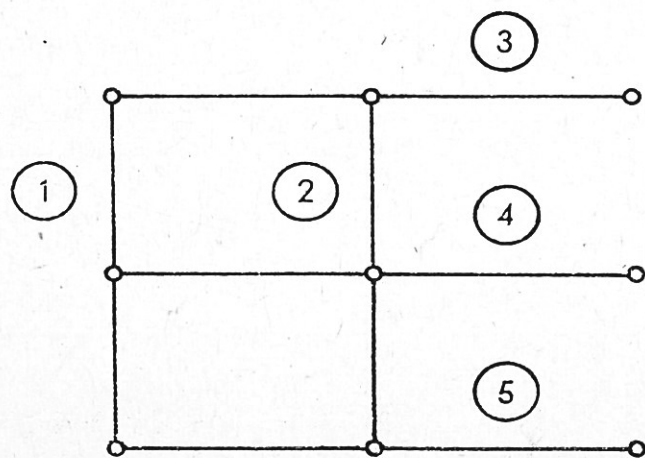
Példaként vizsgáljuk meg a 11. ábrán látható rendszert; itt az egyenessel összekötött ponthármasok az E_i halmazok. A bekarikázott sorszámok a mohó algoritmus által kiválasztott halmazokat mutatják. Látható, hogy ez 3 helyett 5 halmazt választott ki (az első kettőt feleslegesen).

1.17. feladat: Adjunk olyan példát, ahol egy „mohó” algoritmus 11 befedő halmazt választ ki, bár 6 is elég volna.

Bebizonyítjuk most a következőt:

Ha az S -et lefedő E_i halmazok minimális száma k_0 , akkor a „mohó” algoritmus legfeljebb $\frac{11}{6} k_0$ halmazt választ ki (feltéve, hogy minden E_i halmaz három elemű).

A bizonyításhoz némi számolásra lesz szükségünk. Tegyük fel, hogy a „mohó” algoritmus a lépésen keresztül tud diszjunkt,



11. ábra

tehát három újabb pontot lefedő E_i halmazokat választani; utána b lépésen keresztül tud két-két újabb pontot lefedni; végül c lépésen keresztül fed le egy-egy újabb pontot. Ekkor a kiválasztott halmazok száma

$$a + b + c.$$

Nyilvánvaló, hogy az összes pontok száma

$$m = 3a + 2b + c.$$

Mivel k_0 darab E_i halmazzal az összes pont lefedhető,

$$m \leq 3k_0,$$

innen

$$3a + 2b + c \leq 3k_0. \quad (1)$$

Az első lépés után visszamaradó lefedetlen pontok száma $2b + c$. Egy-egy E_i halmaz ezek közül legfeljebb kettőt fed le, tehát ha k_0 darab E_i halmaz mindet lefedi, akkor

$$2b + c \leq 2k_0. \quad (2)$$

Hasonlóan, az első $a + b$ lépés után c darab pont marad lefedetlenül. Egy-egy E_i halmaz ezek közül csak egyet fed le, tehát ha k_0 darab E_i mindet lefedi, akkor

$$c \leq k_0. \quad (3)$$

Szorozzuk meg (1)-et 2-vel, adjuk hozzá (2)-t és (3) háromszorosát, akkor azt kapjuk, hogy

$$6a + 6b + 6c \leq 11k_0,$$

vagyis

$$a + b + c \leq \frac{11}{6} k_0,$$

ahogy állítottuk.

1.18. feladat: Mutassuk meg, hogy 4 elemű halmazok esetén a „mohó” algoritmus a lefedéshez minimálisan szükséges halmazszámnak legfeljebb a $\frac{25}{12}$ -szeresét adja.

1.19. feladat: „Óvatos” algoritmusnak hívhatjuk a következőt: Ha egy E_i halmaz olyan, hogy minden pontját más E_j lefedi, hagyjuk el a rendszerből. Ezt ismételjük, amíg lehet. A visszamaradó fedőrendszer álljon k halmazból, és legyen k_0 az S -et lefedő E_i -k minimális száma. Bizonyítsuk be, ha minden E_i halmaz r elemű, akkor $k/k_0 \leq r$.

1.4.3.

A kromatikus szám

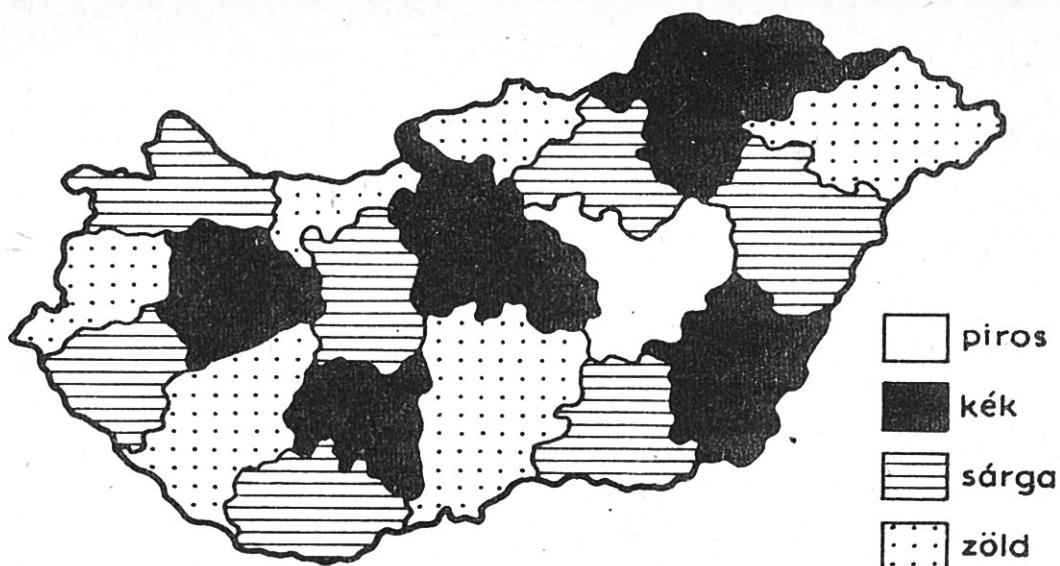
Az eddigiek alapján úgy tűnhet, hogy gyors szuboptimális algoritmusokat találni általában könnyű, gyakran triviális dolog. Ez azonban nem mindig van így; példaként vizsgáljuk meg a gráfok kromatikus számát.

Egy gráf minden csúcsát színezzük ki valamilyen színre. Azt mondjuk, hogy ez a színezés jó, ha bármely él két végpontja különböző színű. Azt a legkisebb számot, ahány színnel a gráf jól kiszínezhető, a gráf kromatikus számának nevezzük. Ez nyilván értelmes definíció, mert ha minden csúcsot más színűre színezünk, jó színezést kapunk, tehát jó színezések léteznek.

A kromatikus szám igen sok alkalmazási és elméleti kérdésben szereplő mennyiség. A 19. században a következő térképszínezési feladattal kapcsolatban vezették be. Ha egy térképen ki akarjuk színezni az országokat, szem előtt kell tartani azt a követelményt, hogy a szomszédos országok különböző színt kapjanak.

Sok tapasztalat alapján fogalmazódott meg az a sejtés — a híres négyszínsejtés —, hogy bármelyik térképet ki lehet négy színnel színezni e követelmény betartása mellett. Ezt a sejtést több, mint 100 év után tudták Appel és Haken bebizonyítani (számítógépek jelentős felhasználásával). Az állítást átfogalmazhatjuk a gráfok nyelvére. Jelöljünk ki minden ország belsejében egy pontot, a „fővárost”, és kössük össze egymást nem metsző vonalakkal a szomszédos országok fővárosait. Ezek a pontok és vonalak egy gráfot alkotnak. Az is megmutatható, hogy minden olyan gráfnak, amely síkban úgy ábrázolható, hogy élei nem metszik egymást (síkba rajzolható), megfelel egy térkép, amiből az a fent leírt módon adódik. E megfeleltetés alapján a négyszín-sejtést így fogalmazhatjuk: Minden síkba rajzolható gráf kromatikus száma legfeljebb 4.

A kromatikus szám meghatározására jó eljárás nem ismeretes; nem tudunk lényegesen jobbat, mint az összes lehetséges „neki-futások” végignézése; hol akadunk el?



Visszatérve a szuboptimális algoritmus kérdéséhez, a kromatikus szám ebből a szempontból is csúnyán viselkedik. *Garey és Johnson* egy meglepő eredménye a következő: Ha létezik olyan algoritmus, mely minden gráfot jól kiszínez úgy, hogy a felhasznált színek száma a kromatikus számnak kevesebb, mint kétszerese, akkor létezik ennél nem lényegesen hosszabb algoritmus a kromatikus szám pontos kiszámítására is. A 2. fejezet eredményei alapján ez a tétel úgy is interpretálható, hogy a kromatikus számot 2-es faktor erejéig effektíven kiszámítani nem lehet.

1.5. Gráfok diagnosztikája

Az algoritmusok hosszára eddig adott alsó korlátaink mind úgy születtek, hogy azt mutattuk meg: már az adatok közül is legalább ennyire volt szükségünk. Amikor a feladat valaminek a kiválasztása (pl. a hamis pénznek, vagy annak, hogy hová kell „beszúrni” az új elemet a listába, vagy a megfelelő sorrendnek az összes lehetséges közül), általában elegendő volt arra hivatkozni, hogy kevesebb adat figyelembevétele nem is nyújthatott volna elég információt.

Sokszor azonban az, amire kíváncsiak vagyunk, egyetlen „igen-nem” kérdés. Van-e a gráfban teljes párosítás? Összefüggő-e a gráf? Prímszám-e egy adott szám? Tegyük fel, hogy egy ilyen kérdés megválaszolására van egy algoritmusunk. Legjobb-e ez? Meg tudjuk-e mutatni azt, hogy minden más algoritmus is legalább ennyi időt igényel?

Próbáljunk meg továbbra is eredeti ötletünkhöz ragaszkodni és azt igazolni, hogy bármely algoritmusnak — a legrosszabb esetben — valamennyi bemenő adatot figyelembe kell vennie. Az egyszerűség kedvéért tegyük fel, hogy a bemenő adatok is „igen-nem” jellegűek, pontosabban, egy nullákból és egyesekből álló sorozat vagy mátrix formájában vannak „kódolva” (mint a gráf adjacenciamátrixa vagy egy szám 2-es számrendszerbeli alakja). Ekkor úgy képzelhetjük el, hogy az algoritmus is „igen-nem” kérdéseket intéz az adatszalaghoz (mintha barkochbázna). Hány ilyen kérdést kell tennie? Az az „információelméleti” megfontolás, melyet korábban használtunk, itt csődöt mond: mivel csak egy „igen-nem” kérdésnyi (szaknyelven: bitnyi) információra van szükségünk, egy kérdés is elég volna, ha éppen azt kérdezhetnénk. De mi csak arra kérdezhetünk, ami közvetlenül adott.

A helyzet az orvosi diagnosztikához hasonló. Az orvos meg akarja állapítani, beteg-e a páciens mája. Nem nézheti meg azonban közvetlenül, hanem csak különféle egyéb vizsgálatok (vér, vizelet stb.) eredményéből következtethet erre. Így ennek az egyetlen kérdésnek az eldöntéséhez sok más kérdésre kell feleletet találnia.

E fejezetben példáinkat gráfok köréből vesszük. Innen a cím: gráfok diagnosztikája. Részletesen tárgyalunk egy módszert, melyel bizonyos körülmények között ki tudjuk mutatni, hogy egy kérdés eldöntéséhez a legrosszabb esetben minden adatra szükség van.

Kezdjük egy igen egyszerű példával. El akarjuk dönteni, hogy egy gráf teljes gráf-e, vagyis össze van-e kötve bármely két csúcsa. Nyilvánvaló, hogy ha a gráf valóban teljes, akkor ehhez valamennyi pontpár összekötöttségét meg kell vizsgálni, hiszen ha egy párt még nem néztünk meg, nem tudhatjuk, hogy össze van-e kötve.

1.5.1. Összefüggőség

Sokkal kevésbé nyilvánvaló példa a gráfok összefüggősége. Megjegyeztük már, hogy ahhoz, hogy egy gráf összefüggőségét eldöntsük, a legrosszabb esetben (akkor, amikor a gráf pontjai két, $\frac{n}{2}$ elemszámú osztályba oszthatók úgy, hogy különböző osztálybeli pontok nincsenek összekötve, de bármely két azonos osztálybeli igen) legalább $\frac{n^2}{4}$ pontpárról meg kellett vizsgálnunk, hogy össze vannak-e kötve. Csak ezután állíthatjuk bizonyosan, hogy a gráf nem összefüggő.

Bebizonyítjuk most, hogy az $\frac{n^2}{4}$ javítható $\binom{n}{2}$ -re. Más szóval:

- **Tétel:** *Bármely olyan algoritmushoz, mely a gráf összefüggő voltát dönti el az adjacenciamátrix ismeretében, van olyan n pontú gráf, melyre az algoritmusnak az összes pontpár összekötöttségét meg kell vizsgálnia.*

(A triviálisan kapott $\frac{n^2}{4}$ -es alsó korláthoz képest ez persze nem nagy élesítés; elvileg érdekes, hogy a helyzet a lehető legrosszabb.)

Képzeljük el, hogy a kérdésről egy barátunkkal vitatkozunk. Ő azt állítja, hogy van olyan algoritmus, mely $\binom{n}{2}$ -nél kevesebb pontpár megvizsgálásával célhoz ér, mi pedig ezt nem hisszük. A vita eldöntésére a következő módszerben állapotunk meg. Ő algoritmusának megfelelően vizsgálja a gráfot; valahányszor egy pontpár összekötöttségét kell tudni, azt tőlem kérdezi meg. Mivel az ő algoritmusának minden gráf esetén működnie kell, én azt felelhetem, amit akarok. Persze igyekszem a számára legkellemetlenebb válaszokat adni. Ha sikerül úgy válaszolnom, hogy csak $\binom{n}{2}$ kérdés — tehát az összes pontpár megvizsgálása — után tudja biztosan megmondani, hogy összefüggő-e a gráf, akkor nekem van igazam; a kérdései és válaszaim nyomán kialakult gráfon az algoritmus $\binom{n}{2}$ pontpár megvizsgálását igényli. Ha ezt nem sikerül elérnem, vagyis $\binom{n}{2} - 1$ lépés után már mindenképpen meg tudja mondani (az utolsónak maradt pár összekötöttségétől függetlenül), hogy a gráf összefüggő-e, akkor algoritmusom valóban olyan jó, mint állította.

Úgy is tekinthetjük az egészet, mint egy barkochbaszerű játékot a barátom és én között. Ő feltehet $\binom{n}{2} - 1$ ilyen kérdést: össze van-e kötve az i -edik és j -edik pont? Én tetszésem szerint válaszolok: igennel vagy nemmel. Ha ő $\binom{n}{2} - 1$ kérdés után meg tudja mondani, hogy összefüggő-e a gráf (a megkérdezetlen éltől függetlenül) ő nyert. Ha a végén ezt még nem lehet megmondani (vagyis a szóba jövő két gráf közül az egyik összefüggő, a másik nem), én nyertem. Ahhoz, hogy bebizonyítsuk a tételt, elég meg-

mutatni, hogy e játékban én nyerek, vagyis megadni a nyerő stratégiámat.

Stratégiám a következő. Nem mondok igent, csak ha „muszáj”. Más szóval, vigyázok arra, hogy a már igenlően megválaszolt és még meg nem kérdezett élek együtt összefüggő gráfot alkossanak; e feltétel mellett azonban, ha tehetem, nemet mondok. Így mindenestre barátom soha nem juthat arra a következtetésre, hogy a gráf — a még meg nem kérdezett élektől függetlenül, tehát még ha azok mind benne volnának is — nem összefüggő.

Tegyük fel, hogy $\binom{n}{2} - 1$ kérdés után biztosan tudná, hogy összefüggő a gráf, vagyis a már igenlően megválaszolt élek összefüggő gráfot alkotnak az utolsó él nélkül is. Legyen (u, v) ez az utolsó, még meg nem kérdezett él. Ekkor tehát speciálisan van egy u -t v -vel összekötő P út, melynek minden élét igenlően választam meg. Ennek pontjait jelölje $u = u_0, u_1, \dots, u_k = v$. Az $(u_0, u_1), (u_1, u_2), \dots, (u_{k-1}, u_k)$ éleket tehát barátom már korábban megkérdezte és én igenlően válaszoltam rá. Stratégiám szerint pl. az (u_0, u_1) párra azért válaszoltam azt, hogy össze van kötve, mert különben a megválaszolatlan és igenlően megválaszolt élek nem alkotnának összefüggő gráfot. De egy lépéssel előbb még ezek az élek összefüggő gráfot alkottak. Ez a gráf tartalmazta az $(u_0, u_1, \dots, u_k, u_0)$ sokszöget. Ha ebből a sokszögből egy élt elhagyunk, könnyű igazolni, hogy még mindig összefüggő marad a gráf. Ezért (u_0, u_1) -re nemlegesen kellett volna felelni! Ez az ellentmondás azt bizonyítja, hogy barátom mégsem tudhatta biztosan megmondani $\binom{n}{2} - 1$ lépés után, hogy összefüggő-e a gráf.

1.20. feladat: Mutassuk meg, hogy minden olyan algoritmus, mely egy n pontú gráfról eldönti, hogy az tartalmaz-e zárt sokszöget, a legrosszabb esetben minden pontpár összekötöttségét meg kell hogy vizsgálja.

1.5.2.

Kieséses verseny és egy rokon feladat

Az előző, viszonylag egyszerű meggondolás után azt várnánk, hogy annak bizonyítása, hogy egy algoritmusnak a legrosszabb esetben az összes rendelkezésre álló adatot fel kell használnia, általában könnyű. Hogy rámutassunk, milyen nehéz is tud lenni ez a kérdés, és milyen finom (talán jelenleg teljes egészükben meg sem értett) különbségeken múlhat az eredmény, vizsgáljuk meg a következő két feladatot:

(a) Turnamentnek hívunk egy olyan irányított gráfot, melyben bármely két pontot pontosan egy él köt össze. Az elnevezés

onnan származik, hogy ha egy teljes körmérkőzést játszik n versenyző (olyan sportágban, ahol döntetlen nincs, pl. pingpongban), akkor az eredményt egy n pontú turnament írja le: az i -edik pontból a j -edikhez nyilat rajzolunk, ha az i -edik versenyző legyőzte a j -ediket.

A feladat: döntsük el, van-e adott turnamentben olyan pont, melyből csak kifelé megy él (vagyis olyan versenyző, aki mindenki mást legyőzött; nevezzük ezért az ilyen pontot abszolút győztesnek).

(b) Adott egy n pontú gráf; döntsük el, hogy van-e olyan pontja, mely minden más ponttal össze van kötve (nevezzük ezt röviden teljes pontnak).

Az (a) és (b) feladatok igen sok szempontból analógok. Pl. akár turnamentről, akár gráfról van szó, az i -edik és j -edik pont viszonyát egyetlen igen-nem kérdéssel tisztázhatjuk. Ennek ellenére a választ adó algoritmusok által megvizsgálandó pontpárok száma egészen máshogyan alakul a két esetben.

A sportvilág már réges-régen talált olyan algoritmust, mely az (a) feladatot megoldja. Játsszanak ugyanis először a versenyzők egy kieséses rendszerű bajnokságot. Ennek során minden mérkőzésben egy versenyző — a vesztes — búcsút mond a versenynek. $n-1$ mérkőzés után tehát csak egyetlen versenyző — a győztes — marad veretlen. Persze nem biztos, hogy ő tényleg mindenkit meg tud verni (lehet, hogy olyasvalakitől kikapna, akit a bajnokság során másvalaki legyőzött). De $n-1$ további mérkőzéssel ezt már el tudjuk dönteni. Így $2n-2$ mérkőzés árán meg tudjuk mondani, van-e abszolút legjobb!

1.21. feladat: Módosítsuk kissé az eljárást úgy, hogy $2n - \lceil \log_2 n \rceil$ pontpár megvizsgálása elegendő legyen.

1.22. feladat: Igazoljuk, hogy ennél kevesebb pontpár vizsgálata nem mindig elegendő. (Útmutatás: játsszunk ismét barátunkkal, mint az előbb. Stratégiánk legyen: ha két veretlen versenyző eredményét kérdezi a barátunk, győzzön az, aki kevesebb veretlen versenyzőt győzött le a korábbiakban; ha az egyik veretlen csak, akkor győzzön ő.

Bebizonyítjuk most: annak eldöntéséhez, van-e a gráfban teljes pont, bármely algoritmus a legrosszabb esetben meg kell hogy vizsgálja valamennyi pontpár összekötöttségét!

Képzeljük el ismét, hogy barátunkkal játszunk: ő ismét $\binom{n}{2} - 1$ olyan kérdést tehet föl, mint: „össze van-e kötve az i -edik és a j -edik pont?”, és utána meg kell tudnia mondani, hogy van-e teljes pont a gráfban.

Nyerő stratégiánk a következő (sokkal ravaszabb, mint az előbb): Az első kérdésre nemlegesen felelünk. Ezek után már nem jöhet minden gráf szóba. A szóba jövők között vannak olya-

nok, melyekben van teljes pont, és vannak olyanok, melyekben nincs. Fogadjuk el egyelőre, hogy az alábbi állítás igaz:

- **Állítás:** Az $x_1, x_2, \dots, x_n$ pontokra rajzolható olyan gráfok száma, melyekben két adott pont (pl. x_1 és x_2) nincs összekötve és melyekben nincs teljes pont, páratlan.

Ekkor tehát első válaszom után a még szóba jövő gráfok közül azoknak a száma, melyekben nincs teljes pont, páratlan. Stratégiám mármost az, hogy mindig úgy válaszolok a kérdésekre, hogy ez a megállapítás érvényben maradjon. Ehhez tudom is tartani magamat. Ugyanis ha egy új kérdést tesznek föl: „Össze van-e kötve a k -adik és l -edik pont?“, akkor az eddig szóba jövő teljes pont nélküli gráfok két osztályra oszlanak aszerint, hogy ez teljesül-e, és én a válaszómmal az egyik osztályt választom ki. Páratlan sok dolgot két osztályba sorolva az egyik osztály szükségszerűen páratlan elemszámú lesz. Tehát tudok úgy választani, hogy a szóba jövő, teljes pontot nem tartalmazó gráfok száma továbbra is páratlan maradjon.

Nézzük a helyzetet a játék végén, $\binom{n}{2} - 1$ kérdés után. A még szóba jövő két gráf közül a teljes pontot nem tartalmazók száma stratégiám szerint páratlan, vagyis 1. Más szóval a két gráf egyike tartalmaz teljes pontot, a másika nem. Tehát én nyertem, mert barátom még nem tudhatja biztosan, van-e teljes pont.

Teljesség kedvéért igazoljuk a fenti „Állítás”-t. Legyenek a pontok $v_1, v_2, \dots, v_n$ és pl. v_1, v_2 az a pár, amelyik nincs összekötve. Azt akarjuk igazolni, hogy azon gráfok száma, melyekben v_1 és v_2 nincs összekötve, és nincs teljes pont, páratlan. E célból a gráfokat osztályokba soroljuk aszerint, hogy a $v_2, \dots, v_n$ pontok milyen gráfot alkotnak. Legyen ez a gráf G' , és legyenek G' -nek v_2 -től különböző teljes pontjai $u_1, \dots, u_k$. Nyilvánvaló, hogy G teljes pontjai az $u_1, \dots, u_k$ pontok közül kell, hogy kikerüljenek; v_1 és v_2 nincsenek összekötve, így nem lehetnek teljesek. Így ha azt akarjuk, hogy G -ben ne legyen teljes pont, v_1 -et nem szabad $u_1, \dots, u_k$ -val és (a kikötések szerint) v_2 -vel összekötni; a maradék $(n-1) - k - 1$ ponttal azonban úgy köthetjük össze v_1 -et, ahogy csak akarjuk. Így G' -ből pontosan $2^{n-1-k-1}$ olyan G gráfot származtathatunk, melyben nincs teljes pont és v_1, v_2 nincsenek összekötve. Ez a szám páros, kivéve ha $k = n - 2$. Ez azonban abban az esetben van csak így, amikor G' teljes gráf.

Tehát egyetlenegy G' gráf páratlan számú, a többi pedig páros számú G gráfot szolgáltat. Ezért a G gráfok száma páratlan.

1.23. feladat: Mi a párossága az összes n pontú, teljes pontot nem tartalmazó gráfok számának?

1.5.3.

A probléma általánosítása

Most általánosabban meghatározzuk az olyan feladatok körét, melyekre vonatkozóan a fentihez hasonló vizsgálatok elvégezhetők. Jelöljük A_{ij} -vel azt az állítást, hogy az i -edik és j -edik pont össze van kötve egy adott, mondjuk 4 pontú G gráfban (elegendő az $i < j$ esetet nézni). Jelölje $\wedge$, ill. $\vee$ az „és”, ill. a „vagy” kötőszókat. Ekkor

$$A_{12} \wedge A_{13} \wedge A_{14}$$

azt jelenti, hogy az 1-es pont teljes. Így

$$(A_{12} \wedge A_{13} \wedge A_{14}) \vee (A_{12} \wedge A_{23} \wedge A_{24}) \vee (A_{13} \wedge A_{23} \wedge A_{34}) \vee (A_{14} \wedge A_{24} \wedge A_{34}) \quad (1)$$

azt jelenti, hogy van a gráfban teljes pont. Legyen adva másrészt egy 4 pontú turnament. Jelölje B_{ij} azt a kijelentést, hogy az i -edik versenyző legyőzte a j -ediket ($i < j$). Jelölje $\bar{B}_{ij}$ azt, hogy a j -edik versenyző győzte le az i -ediket. Ekkor

$$(B_{12} \wedge B_{13} \wedge B_{14}) \vee (\bar{B}_{12} \wedge B_{23} \wedge B_{24}) \vee (\bar{B}_{13} \wedge \bar{B}_{23} \wedge B_{34}) \vee (\bar{B}_{14} \wedge \bar{B}_{24} \wedge \bar{B}_{34}) \quad (2)$$

azt fejezi ki, hogy van abszolút győztes. Mindkét esetben tehát azt szeretnénk meghatározni, igaz-e a felírt állítás; és azt vizsgáljuk, hány ilyen kérdést kell feltennünk: „igaz-e A_{ij} ?”, ill. „igaz-e B_{ij} ?”.

Általánosan így fogalmazhatjuk meg a feladatunkat: vegyünk n egyelőre ismeretlen állítást, $x_1, \dots, x_n$ -t. Építsünk föl ezekből az $\wedge$ (és), $\vee$ (vagy) és fölérzés (tagadás) segítségével egy kifejezést, ún. logikai függvényt, pl. a következőt:

$$((x_1 \wedge x_2) \vee \bar{x}_3) \wedge (\bar{x}_1 \wedge x_2) \wedge x_4.$$

Jelöljük ezt $f(x_1, \dots, x_n)$ -nel. Ha minden x_i -ről tudjuk, hogy igaz-e vagy sem, akkor tudjuk $f(x_1, \dots, x_n)$ -ről is. Most azonban tegyük föl, hogy $x_1, \dots, x_n$ igaz vagy hamis voltát (logikai értékét) eldönteni költséges (pl. kísérletet kell végeznünk), s így olyan eljárást szeretnénk adni, amely úgy dönti el $f(x_1, \dots, x_n)$ logikai értékét, hogy ehhez csak lehetőleg kevés x_i igazságát (logikai értékét) kell meghatározni.

Mint láttuk, a (2) típusú logikai függvény értéke meghatározható úgy, hogy csak $2n-2$ változójának az igaz vagy hamis voltát kell meghatározni, míg az (1) logikai függvény esetében valamennyi változóra szükség lehet. Mi lehet a különbség a két

eset között? Hiszen nagyon hasonló alakú kifejezésekről van szó! Erre jelenleg csak egy sejtés (tehát bizonyítatlan állítás) vonatkozik. Ennek megfogalmazása végett mondjuk azt, hogy $f(x_1, \dots, x_n)$ változói egyenértékűek, ha bármely x_1 változót tekintve lehet a függvény változóit úgy átrendezni (permutálni), hogy x_1 kerüljön az első helyre és a függvény ugyanaz maradjon. Pl. az $(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ logikai függvény változói egyenértékűek, mert

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) = (x_2 \wedge x_1) \vee (x_3 \wedge x_4) = (x_3 \wedge x_4) \vee (x_1 \wedge x_2) = \\ = (x_4 \wedge x_3) \vee (x_1 \wedge x_2).$$

De az $x_1 \vee \bar{x}_2$ polinom változói nem egyenértékűek, mert $x_1 \vee \bar{x}_2$ nem mindig egyenlő $(x_2 \vee \bar{x}_1)$ -gyel.

A sejtés, az ún. Aanderaa—Rosenberg-sejtés, így szól:

Ha az $f(x_1, \dots, x_n)$ függvényben nincs tagadásjel és változói egyenértékűek; akkor bármely olyan algoritmus, amely $f(x_1, \dots, x_n)$ értékét kiszámítja, a legrosszabb esetben valamennyi változó értékét meg kell, hogy határozza.

Az (1) függvényre ez igaz, bizonyítottuk. Több speciális esetben a fenti sejtést bizonyították (*Best, Schrijver, van Emde-Boas, Rivest, Vuillemin* és mások munkája nyomán). Ezek közül kettő egyszerűen megfogalmazható: I) ha n prímszám; II) ha a változók egyenértékűségét az biztosítja, hogy az $x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_n \rightarrow x_1$ változócsere sem változtatja meg a függvényt.

1.24. feladat: Az

$$(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4)$$

függvényben nincs tagadásjel. Mutassuk meg, hogy logikai értéke mégis kiszámítható úgy, hogy csak három változó logikai értékére van szükség.

1.25. feladat: Az

$$(x_1 \wedge \bar{x}_2 \wedge \bar{y}_1 \wedge y_2) \vee (x_2 \wedge \bar{x}_3 \wedge \bar{y}_2 \wedge y_3) \vee (x_3 \wedge \bar{x}_1 \wedge \bar{y}_3 \wedge y_1)$$

függvényben minden változó egyenértékű. Mutassuk meg, hogy logikai értéke mégis kiszámítható úgy, hogy csak 5 változó logikai értékére van szükség.

2. fejezet

Kereső feladatok

Az előző fejezetben sokféle problémát vizsgáltunk és különféle algoritmusokat adtunk rájuk. Az algoritmusok hossza (lépésszámigénye) igen széles keretek között változott: n paraméter esetén néha $\log_2 n$, néha 2^n volt a lépésszámigény. Ez a meg lehetőségen kaotikus helyzet sürgetően követeli az egységesítő elméletet (ugyanakkor persze megalkotásának nehézségére is rámutat). A „Gráfok diagnosztikája” c. részben említettünk példát arra, hogy néhány feladatra egységes megfogalmazás található és — legalábbis sejtés erejéig — még meg lehetőségen általános kijelentést is meg tudtunk fogalmazni az őket megoldó algoritmusokkal kapcsolatban. Az ott tárgyaltaknak azonban van egy alapvető hiányossága: a módszer nem képes figyelembe venni, hogy mi történik a számítógép belsejében. Megkerüli ezt a kérdést azzal, hogy csak azt vizsgálja, hányszor nyúl a gép az adatszalaghoz. Ezért aztán elvileg képtelen többet mondani, mint hogy minden

adatra szükség van, vagyis, hogy a gépnek legalább annyi lépést kell dolgoznia, mint az adatok száma. Láttuk ugyanakkor, hogy vannak olyan feladatok (pl. a pakolási probléma), amikor az n^2 számú bemenő adaton minden ismert algoritmus exponenciálisan hosszú ideig dolgozik. Ilyen konklúzióra a fenti elmélet nem juthat.

2.1.

A problémák problémája

2.1.1.

Ahol egy kismadár is segíthet

Ebben a fejezetben egy olyan elmélet alapjait ismertetjük, amely megpróbál ezzel a nehézséggel megbirkózni. Induljunk ki néhány példából:

1. **példa:** Szorozzunk össze két n jegyű számot!
2. **példa** (pakolási probléma): Keressünk egy G gráfban maximális független ponthalmazt!
3. **példa** (színezési probléma): Színezzük ki „jól” a G gráf csúcsait 4 színnel (ha lehet)!
4. **példa** (prímprobléma): Találjunk egy adott n jegyű számhoz valódi osztót, ha van neki (tehát nem prímszám)!
5. **példa** (párosítási probléma): Adott páros gráfban keressünk olyan független érendszert, amely az összes „alsó” pontot tartalmazza (ha van ilyen)!

E példák között érdekes különbség tételére ad lehetőséget az alábbi gondolat. Tegyük fel, hogy vállunkon egy tudós kismadár ül, aki valami rövidet a fülünkbe súghat. A 3., 4. és 5. példa esetén ez a kismadár egyszerűen megoldhatja a feladatunkat azzal, hogy megsúg valamit: így a harmadik feladatnál azt, hogy milyen színre színezzük az egyes pontokat, a negyediknél az adott szám egy osztóját, az ötödiknél a keresett párosítást. Ha már megsúgta, szinte nincs mit csinálnunk; annak ellenőrzése, hogy jó is, amit súgott (sose bízhatunk egy madárban), gyerekjáték.

Kicsit más a helyzet az 1. és 2. feladat esetében: itt is megsúghatja a kismadár az eredményt, de ha nem bízunk benne, nem segített semmit. Annak ellenőrzésére, hogy két szám szorzata valóban a megsúgott szám-e, nemigen van más mód, mint összeszorozni őket; annak ellenőrzése, hogy az általa megsúgott pontok halmaza maximális független halmaz, nemigen történhet más-

képpen, mint az összes független halmazok többé-kevésbé ügyes végigvizsgálásával. (Az, hogy a halmaz tényleg független-e, persze könnyen ellenőrizhető.)

Kereső feladatnak: nevezzük az olyan feladatokat, amelyekben valamit keresünk, és ha a kismadár (vagy a szerencse vagy az intuíció) a keresett dolgot megsúgja, ezáltal a kérdés triviálissá válik (később pontosabb meghatározást is adunk).

A kereső feladatoknál persze nem tudhatjuk, hogy létezik-e egyáltalán az a valami, amit keresünk. Ha nem, akkor a kismadár nem tud mit sugni. Hiába mondja azt, hogy nincs ilyen dolog, ennek ellenőrzése azonos nehézségű az eredeti feladattal.

5. példánkban azonban — ha a kismadár ismeri a Hall-tételt — (1.2.3. pont), akkor ilyenkor is tud segíteni. Ha nincsen kívánt tulajdonságú független élrendszer, akkor meg tud sugni egy „alsó” pontokból álló halmazt, melynek kevesebb szomszédja van, mint eleme. Ennek helyességét ismét könnyen ellenőrizhetjük.

Valójában arról van szó, hogy az 5. feladattal párhuzamosan megfogalmazhatunk egy másik kereső feladatot:

5.* példa: Válasszuk ki adott páros gráf „alsó” pontjainak olyan részalmazát, melynek kevesebb szomszédja van, mint eleme.

Az 5. és 5.* kereső feladatok *komplementáris* (kiegészítő) feladatok abban az értelemben, hogy kettőjük közül mindig pontosan egynek van megoldása (ezt mondja ki Hall tétele).

Érdemes megjegyezni, hogy a 4. példában tárgyalt feladatnak, a prímség eldöntésének (vagyis valódi osztó keresésének) is van komplementáris feladata. Ennek a megfogalmazása azonban bonyolult számelméleti eszközöket igényel, így itt eltekintünk tőle. A 3. feladatnak, a gráfszínezésnek eddig nem találták komplementáris feladatát (valószínűleg nincs is).

A 2. feladat ugyan a mondott formában nem kereső feladat, de visszavezethető olyanra. Legyen ugyanis a gráfnak n pontja, és tekintsük $k=2, \dots, n$ -re a következő feladatokat:

2.* példa: Keressünk az adott G gráfban k független pontot!

Ha ezt a feladatot minden k -ra megoldottuk, akkor a legnagyobb olyan k -t tekintve, melyre ilyen halmazt találtunk, megvan a keresett maximális független halmaz.

Megjegyzendő, hogy a 2.* feladatnak sem ismert komplementáris feladata.

Itt az ideje, hogy pontosabban definiáljuk a kereső feladat fogalmát. Minden kereső feladatban van egy adathalmaz, vala-

milyen alapul vett struktúra (gráf a 2*. és 3., egész szám a 4., páros gráf az 5. példában). Ezt a struktúrát vagy adathalmazt egy 0—1 sorozat formájában kódolva képzeljük (egész szám esetén 2-es számrendszerbeli alakjában, gráf esetén az adjacenciamátrixa segítségével stb.). Ennek a sorozatnak a hosszát nevezzük a konkrét feladat *méretének*. Minden esetben keresünk valamit; ezt a valamit célszerűen ugyancsak 0—1 sorozat formájában képzeljük el.

Mint mondtuk, a kereső feladatok alaptulajdonsága az, hogy bármely 0—1 sorozatról (melyet a kismadár fülünkbe csivitel) könnyen el tudjuk dönteni, hogy megfelelő-e. Mit jelent az, hogy „könnyű”? Itt különböző definíciókat alapul véve, a kereső feladat fogalmának különböző változataihoz jutunk. Ezek közül egy bizonyult különösen gyümölcsözőnek, és mi itt csak ezt az egyet tárgyaljuk: eszerint „könnyen” azt jelenti, hogy adható olyan algoritmus, amely minden n méretű adatsorozat és minden „találgatás” esetén legfeljebb n^{konstans} lépésben megmondja, hogy helyesen „tippeltünk-e” (természetes, hogy a „találgatás”-ként beadható 0—1 sorozat hossza is legfeljebb n^{konstans}). Azt az algoritmust, amelyhez van olyan c szám, hogy n számú bemenő adat esetén az algoritmus lépésszáma nem nagyobb, mint n^c , *polinomiális algoritmusnak* fogjuk nevezni. Tehát a „könnyen”-t úgy értjük, hogy „polinomiális algoritmussal”.

Az 5. példában megfogalmazott kereső feladat polinomiális algoritmussal megoldható; a 2* és a 3. példa annyit legalábbis mutat, hogy ilyen algoritmus létezése nem magától értetődő, sőt, valószínűleg nem is mindig létezik. A továbbiakban egy Cook, Karp és Levin által kidolgozott, erre vonatkozó elméletet ismertetünk.

2.1.2.

Egy univerzális feladat: a kielégíthetőség

A „Gráfok diagnosztikája” c. részben a logikai függvények segítségével az ott tárgyalt problémának általános megfogalmazást tudtunk adni. Hasonlóan fogunk itt is eljárni. Most azonban csak bizonyos speciális logikai függvényekre lesz szükségünk. Jelöljenek $A_1, \dots, A_n$ bizonyos állításokat. Ezekkel más állításokat már kifejezhetünk. Emlékeztetünk arra, hogy az „és” szócskát a $\wedge$ jellel, a „vagy” szócskát a $\vee$ jellel, a tagadást fölhúzással jelöljük. Ekkor pl.

$$A_1 \vee A_2 \vee \overline{A_3}$$

azt fejezi ki, hogy vagy A_1 , A_2 közül legalább egy igaz, vagy A_3 nem teljesül. Nevezzük *elemi összegnek* az olyan formulákat, me-

lyeket úgy kapunk, hogy az A_i tulajdonságok és tagadásaik közül bizonyosakat $\vee$ (vagy) jellel összekapcsolunk. Pl.

$$A_1 \vee \bar{A}_2 \vee \bar{A}_3 \vee A_4$$

elemi összeg, de

$$A_1 \vee (\bar{A}_2 \wedge \bar{A}_3), \quad A_1 \vee (\overline{A_2 \vee A_3})$$

nem. Ha több ilyen elemi összeget az $\wedge$ (és) jellel kapcsolunk össze, akkor egy újabb formulát kapunk, mely akkor és csak akkor igaz, ha a tekintett elemi összegek mindegyike igaz. Pl.:

$$(A_1 \vee A_2) \wedge (\bar{A}_3 \vee A_4) \wedge (\bar{A}_1 \vee A_3 \vee A_4) \wedge (A_2 \vee \bar{A}_4).$$

Az ilyen kifejezést *konjunktív normálformának* vagy röviden *normálformának* hívjuk. Az $A_1, \dots, A_n$ állítások igaz vagy hamis voltától függően a konjunktív normálforma értéke is lehet igaz vagy hamis. Az $A_1, \dots, A_n$ -et a normálforma változóinak nevezzük, az előállításában szereplő elemi összegeket *tényezőknek*, számukat pedig a normálforma *hosszának*.

Megjegyezzük, hogy minden olyan állításfüggvény, amelynek igaz volta az $A_1, \dots, A_n$ állítások igaz voltától függ, előállítható, mint ezek normálformája. Például az, hogy „ A_1, A_2, A_3, A_4 közül legalább kettő igaz”, így írható:

$$(A_1 \vee A_2 \vee A_3) \wedge (A_1 \vee A_2 \vee A_4) \wedge (A_1 \vee A_3 \vee A_4) \wedge (A_2 \vee A_3 \vee A_4).$$

Egy ilyen előállítás hossza általában igen nagy lehet. Mondhatnánk, hogy azok a „nagyon egyszerű” tulajdonságok, melyeket rövid normálformával elő lehet állítani.

Ez azonban félrevezető lehet: pl. azt a tényt, hogy „az $A_1, \dots, A_{2n+1}$ állítások több, mint fele igaz”, csak $\binom{2n+1}{n} \approx \frac{2 \cdot 4^n}{\sqrt{\pi n}}$ hosszúságú normálformával lehet felírni.

2.1. feladat: Adjunk meg egy ilyen normálformát és bizonyítsuk be, hogy rövidebb nem fejezheti ki ugyanazt!

A rövid normálformákkal kapcsolatban vetődik fel fő problémánk. Egy normálformát *kielégíthetőnek* mondunk, ha változóinak van olyan logikai értéke, melyre igaz. Például

$$(A_1 \vee A_2) \wedge \bar{A}_1$$

kielégíthető: ha A_1 hamis, A_2 pedig igaz, akkor a normálforma logikai értéke igaz. De az

$$(A_1 \vee A_2) \wedge \bar{A}_1 \wedge \bar{A}_2$$

nem kielégíthető: ha A_1 és A_2 is hamis, akkor az első tényező, ha van köztük igaz, akkor a második vagy harmadik tényező nem igaz.

Most már megfogalmazhatjuk a következő fontos problémát:

6. példa: Adott egy konjunktív normálforma. Döntsük el, hogy kielégíthető-e?

Vegyük észre a következőt. Ha az $A_1, \dots, A_n$ változók értékeit tudjuk (egy kismadár megsúgta), akkor a normálforma értékét könnyű megmondani: ha minden tényezőjében van egy igaz tag (tehát egy igaz A_i vagy egy hamis A_i föléhúzva), akkor az értéke igaz; egyébként hamis. Itt tehát ismét egy kereső feladatunk van: az $A_1, \dots, A_n$ változóknak olyan értékét kell keresni, melyre az adott normálforma igaz.

Kereső feladatunk méretként pl. a normálformában előforduló változójelek számát tekinthetjük. Így, ha n változó van és m tényező, akkor a méret legfeljebb mn . (Ha a formulát 0—1 sorozatként kódoljuk, akkor a sorozat ennél hosszabb lesz, hiszen pl. egy változójel kódolásához több 0-ra, ill. 1-esre van szükség. Ezt azonban elhanyagolhatjuk.)

Ha a normálforma változóinak száma kicsi a méretéhez képest (pl. ha $n \approx \log m$), akkor az összes lehetséges logikai értékrendszerek végigvizsgálása elég hatékornak tekinthető. De ha pl. $n \approx m$, akkor ez már exponenciálisan sok lépést igényel a feladat méretéhez képest, és ezért más módszerek után kell nézni.

A 6. példának az a jelentősége, hogy *univerzális kereső feladat*, vagyis minden más kereső feladat visszavezethető rá.

Ezt a 3. és 2.* feladat példáján mutatjuk be. Az általános bizonyítás az algoritmus fogalmának teljes precizitását igényli és ezért kissé hosszadalmas. Tekintsük először a gráfok 3 színnel való színezhetőségének kérdését.

Jelölje P_i , K_i , ill. Z_i azt az állítást, hogy a G gráf i -edik pontja piros, kék, ill. zöld. Ekkor írjuk fel, hogy

$$(\bar{P}_i \vee \bar{P}_j) \wedge (\bar{K}_i \vee \bar{K}_j) \wedge (\bar{Z}_i \vee \bar{Z}_j) \quad (1)$$

minden olyan i, j párra, melyre az i -edik és j -edik pont össze van kötve. Ez azt fejezi ki, hogy a szomszédos pontok színe nem lehet azonos. Írjuk fel továbbá minden i -re, hogy

$$P_i \vee K_i \vee Z_i, \quad (2)$$

vagyis hogy az i -edik pontnak van valamilyen színe; továbbá azt, hogy

$$(\bar{P}_i \vee \bar{Z}_i) \wedge (\bar{Z}_i \vee \bar{K}_i) \wedge (\bar{K}_i \vee \bar{P}_i), \quad (3)$$

vagyis hogy csak egyetlen színe van.

Ha az (1), (2) és (3) kifejezéseket $\wedge$ (és) jelekkel összekapcsoljuk, az eredményül kapott kifejezés azt mondja, hogy a P_i , K_i , Z_i

állítások a gráfnak egy jó színezését írják le. Ez a normálforma (melyben P_i, K_i, Z_i változók) pontosan akkor lesz kielégíthető, ha a gráfnak van jó színezése.

Mivel az így kapott kifejezés hossza n pontú gráf esetén $2n^2$ -nél biztosan nem több, minden olyan algoritmus, mely polinomiálisan megoldja a 6. példában megfogalmazott problémát, a gráfok 3 színnel való színezhetőségére is választ ad. Ebben az értelemben tekinthetjük a 3. példát a 6. példára *visszavezethetőnek*.

Nézzük a 2.* példát: adott egy G gráf és egy k természetes szám, keressük meg G -nek egy k elemű független pontthalmazát!

Tegyük fel, hogy kiválasztottunk egy k elemű független S halmazt. Sorszámozzuk meg S elemeit, és jelölje B_{ij} az alábbi állítást: „A gráf i -edik pontja S -nek j -edik pontja.” ($1 \leq i \leq n$, $1 \leq j \leq k$). Ekkor a következők teljesülnek:

$$B_{1j} \vee B_{2j} \vee \dots \vee B_{nj} \quad (4)$$

minden $1 \leq j \leq k$ esetén (vagyis S -nek van j -edik eleme):

$$\bar{B}_{i_1j} \vee \bar{B}_{i_2j} \quad (5)$$

minden $1 \leq j \leq k$, $1 \leq i_1 < i_2 \leq n$ -re (vagyis S -nek csak egy j -edik eleme van),

$$\bar{B}_{ij_1} \vee \bar{B}_{ij_2} \quad (6)$$

minden $1 \leq i \leq n$, $1 \leq j_1 < j_2 \leq k$ esetén (vagyis S elemei különbözőek), és

$$\bar{B}_{i_1j_1} \vee \bar{B}_{i_2j_2} \quad (7)$$

minden olyan $1 \leq i_1 < i_2 \leq n$, $1 \leq j_1 < j_2 \leq k$ indexrendszerre, melyre az i_1 -edik és i_2 -edik pont egymással össze van kötve. (Ez azt jelenti, hogy S független pontthalmaz). Ha az összes (4), (5), (6) és (7) alatti kifejezéseket az $\wedge$ jellel összekapcsoljuk, egy olyan K kifejezést kapunk, amely a B_{ij} állítások normálformája és amely azt fejezi ki, hogy az S egy k elemű független halmaz. Megfordítva, ha a B_{ij} -ket változóknak tekintjük, a K normálforma akkor és csak akkor elégíthető ki, ha gráfunkban van k elemű független halmaz.

Észrevehetjük, hogy a K normálforma hossza kisebb, mint n^4 . Így beláttuk, hogy a pakolási probléma is visszavezethető a normálformák kielégíthetőségének kérdésére.

A két tárgyalt visszavezetés alapján talán érezhető, hogy bármely kereső feladat visszavezethető a normálformákra vonatkozó kielégíthetőségi problémára.

- **Tétel:** *A kielégíthetőség problémája univerzális kereső feladat, vagyis minden kereső feladat visszavezethető rá.*

2.2. Nehéz kombinatorikai feladatok

Mivel a konjunktív normálforma fogalma igen általános logikai fogalom, nem meglepő, hogy segítségével megadható a kereső feladatok közös megfogalmazása. Meglepőbb tény, hogy más univerzális kereső feladatok is vannak, mégpedig igen egyszerűek: pl. a pakolási, a gráfszínezési, a fedési és a hátizsák-probléma. Ezt megmutatjuk az első kettőről. Módszerünk most az lesz, hogy a normálformák kielégíthetőségi kérdését vezettjük vissza ezekre.

2.2.1. Közvetlen átfogalmazás

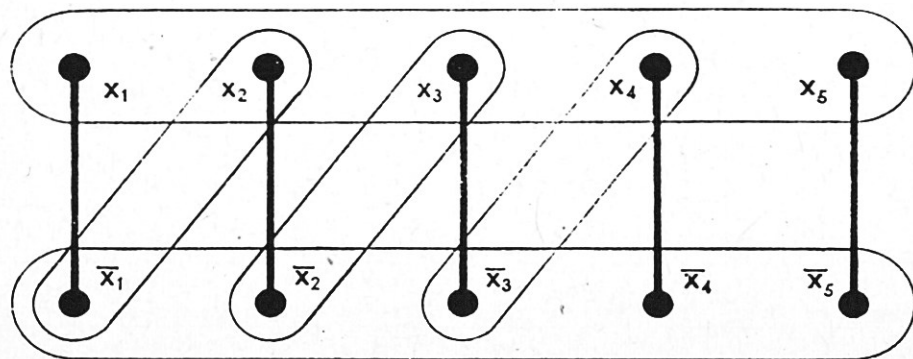
Legyen tehát adva egy K konjunktív normálforma; konstruálni fogunk egy olyan G gráfot, amely akkor és csak akkor színezhető ki 3 színnel, ha K kielégíthető. A konstrukciót az

$$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee x_4) \wedge (\bar{x}_4 \vee x_5) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5) \quad (8)$$

forma példáján mutatjuk be, de elvégezhető volna ugyanígy bármely más konjunktív normálforma esetén is.

Ehhez a logikai feladatot kombinatorikai feladattá fogalmazzuk át. Vegyünk fel 10 pontot, és betűzzük meg őket az $x_1, x_2, x_3, x_4, x_5, \bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5$ betűkkel. Az adott (8) kifejezés minden tényezőjéhez képezzük azon pontok halmazát, melyek jele a tényezőben fellep. Kerítsük körül ezeket a pontokat (12. ábra). Kössük össze továbbá az $(x_i, \bar{x}_i)$ pontpárt egy éllel.

Mármost, ha az x_i -k logikai értéke olyan, hogy (8) igaz, akkor tekintsük mindazokat a pontokat, melyek „jele” igaz (tehát x_i -t, ha x_i igaz és $\bar{x}_i$ -t, ha x_i nem igaz). Ezekről a következőket állapítjuk meg:



12. ábra

- (a) számuk annyi, mint a változóké (esetünkben 5);
- (b) bármely $(x_i, \bar{x}_i)$ párból és bármely körülkerített halmazból tartalmaznak elemet.

A példánkon az $\{x_1, x_2, x_3, x_4, \bar{x}_5\}$ halmaz ilyen tulajdonságú.

Megfordítva, ha ki tudjuk választani az adott pontoknak egy olyan részhalmazát, melyre (a) és (b) teljesül, akkor a (8) kifejezés kielégíthető.

2.2. feladat: Mutassuk meg, hogy ezzel a kielégíthetőségi feladatot lényegében a fedési feladatra (1.4.3. pont) vezettük vissza.

2.2.2.

A háromszín-probléma

Most olyan gráfot konstruálunk, melynek 3 színnel való színezhetősége egyenértékű lesz az eredeti (8) normálforma kielégíthetőségével. Ez hosszabb, bonyolultabb és kevésbé „természetes” konstrukció lesz, mint az előbbi volt.

- **Tétel:** *A normálformák kielégíthetőségének feladata visszavezethető a gráfok 3 színnel színezhetőségének feladatára, tehát az utóbbi is univerzális kereső feladat.*

Ennek bizonyításához abból a konstrukcióból indulunk ki, amit az 1.2.1. pontban a (8) kifejezésnél használtunk és a 12. ábrán látható. Felvesszük tehát az $x_1, x_2, \dots, \bar{x}_1, \bar{x}_2, \dots$ pontokat, az x_i és $\bar{x}_i$ pontokat éllel összekötjük, és körülkerítjük a normálforma egy-egy tényezőjében előforduló pontok halmazait.

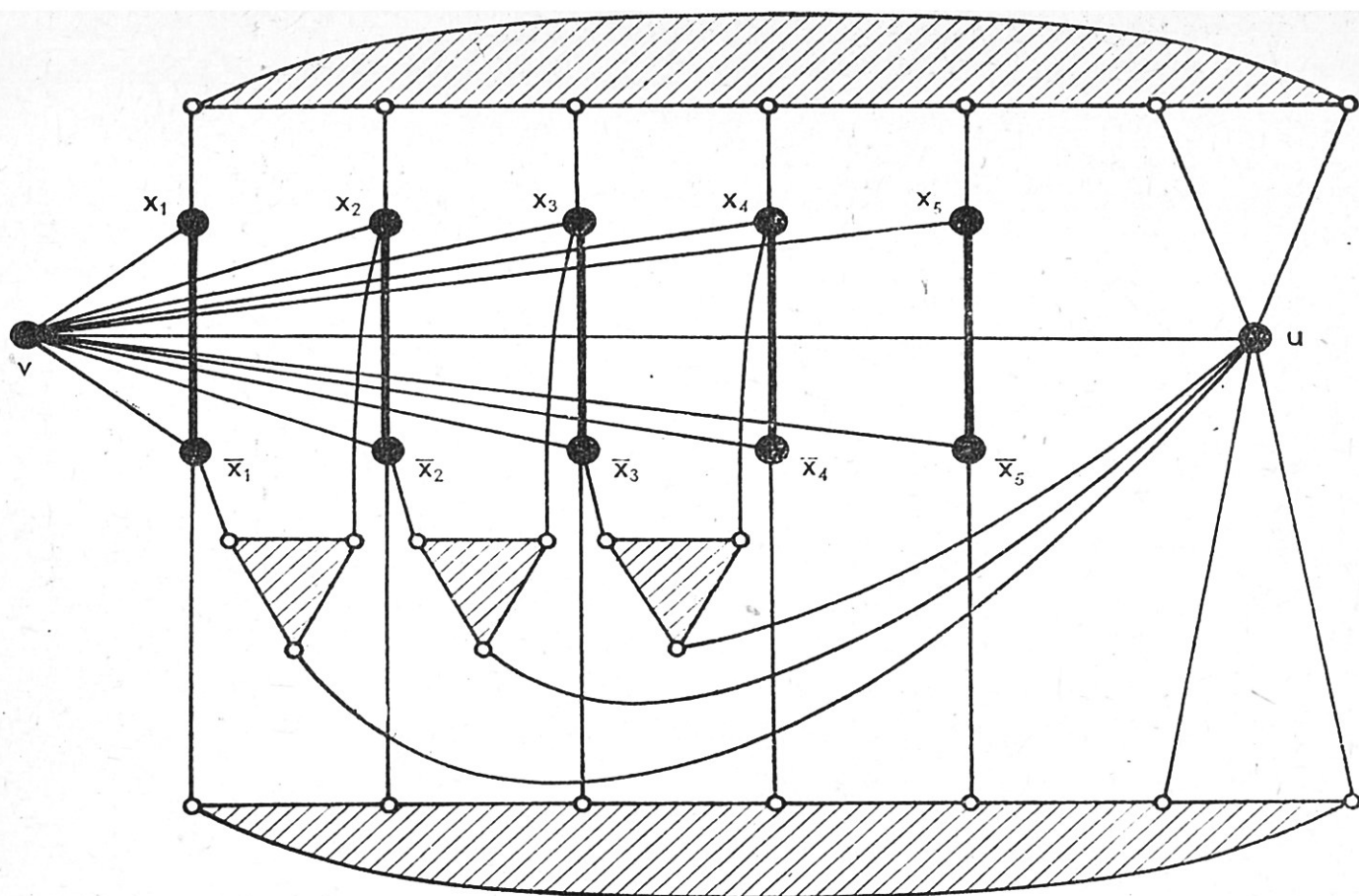
Vegyünk fel továbbá két új pontot, u -t és v -t. Kössük össze ezeket egymással, továbbá v -t minden $x_i, \bar{x}_i$ ponttal. Tartsuk meg az $(x_i, \bar{x}_i)$ éleket.

Tekintsünk egy körülkerített halmazt. Ha ez páratlan elemszámú, mint például $\{x_1, x_2, x_3, x_4, x_5\}$, akkor vegyünk fel egy kettővel több csúcsú sokszöget, és kössük össze két csúcsát u -val, a többit pedig rendre a körülkerített pontokkal.

Ha a körülkerített halmaz páros elemszámú, mint pl. $\{\bar{x}_1, x_2\}$, akkor egy eggyel több csúcsú (tehát ismét páratlan) sokszöget veszünk fel, egy csúcsát u -val, a többit rendre a körülkerített pontokkal kötjük össze. Így tehát egy G gráfot szerkesztettünk (13. ábra).

- **Állítás:** *A G gráf akkor és csak akkor színezhető 3 színnel, ha az eredeti problémánkra a válasz igenlő.*

Tegyük fel először, hogy G kiszínezhető 3 színnel, mondjuk pirossal, kékkel és zölddel. Legyen pl. u kék, v pedig zöld. Ekkor



13. ábra

az eredeti pontok között zöld színű nem lehet, mert mind össze van kötve v -vel. Ezért az $(x_i, \bar{x}_i)$ párból — melyeket él köt össze — az egyik piros, a másik kék.

Másrészt vegyük észre, hogy nem lehet egyetlen körülkerített halmaznak sem minden eleme kék. Ez ugyanis azt eredményezné, hogy a hozzá tartozó páratlan sokszögben kék pont nem fordulhatna elő, hiszen annak minden csúcsa vagy u -val, vagy egy körülkerített ponttal össze van kötve. Márpedig két színnel egy páratlan sokszög nem színezhető ki. Ez az ellentmondás bizonyítja, hogy nem lehet egyetlen körülkerített halmaznak sem minden eleme kék színű.

Tehát a kék színű pontok olyanok, hogy minden $(x_i, \bar{x}_i)$ párból az egyik közéjük tartozik, de nem tartozik egyetlen körülkerített halmaznak sem minden eleme közéjük. Így a piros pontok kielégítik az (a) és (b) feltételeket.

Megfordítva, tegyük fel most, hogy van a pontoknak olyan halmaza, mely eleget tesz az (a) és (b) feltételeknek. Ebből kiindulva megszínezzük G -t 3 színnel.

Az adott halmaz pontjait színezzük pirosra, az $x_i, \bar{x}_i$ pontok közül a többit kékre, u -t kékre, v -t zöldre. Azt kell megmutatnunk, hogy ez a színezés [mely az $(x_i, \bar{x}_i)$ és a v -ből kiinduló éleken nyilván jó] kiterjeszthető a legutoljára felvett páratlan sokszögekre.

Legyen pl. $(x_1, x_2, \dots, x_{2k})$ egy körülkerített halmaz, és $(y_1, \dots, y_{2k+1})$ a hozzá tartozó páratlan sokszög (a páratlan elemszámú körülkerített halmazokhoz tartozó sokszögre a kiterjesztés hasonlóan tehető meg). Az $x_1, \dots, x_{2k}$ pontok között (b) feltevésünk szerint van piros, legyen a legelső piros x_i , és tegyük fel, hogy $i \geq 2$. Színezzük y_i -t kékre, majd színezzük ki az $y_{i+1}, \dots, y_{2k+1}, y_1, \dots, y_{i-1}$ pontokat egymás után. Minden esetben csak két szín van kizárva; az előző y -é és a szomszédos x (vagy u) ponté. Így mindig tudunk megfelelő színt találni. Csak abból lehetne baj, ha a végén y_{i-1} és y_i azonos színűek lennének. De y_{i-1} színe nem kék, mert x_{i-1} kék; és y_i is kék, tehát ez a nehézség nem lép fel.

Az $i=1$ eset teljesen hasonlóan intézhető el. Ezzel beláttuk, hogy G akkor és csak akkor színezhető ki három színnel, ha az (a), (b) feltételeket kielégítő halmaz létezik.

Bebizonyítottuk tehát, hogy egy gráf 3 színnel való színezhetősége univerzális kereső feladat.

2.3. feladat: Adjunk polinomiális algoritmust annak eldöntésére, hogy egy gráf kiszínezhető-e két színnel

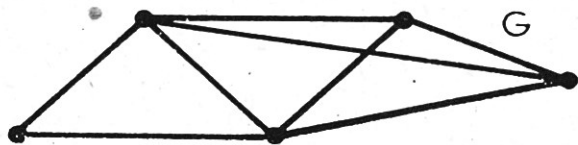
2.4. feladat: Igazoljuk, hogy minden $k \geq 3$ -ra annak eldöntése, hogy egy gráf kiszínezhető-e k színnel, univerzális kereső feladat.

2.2.3.

A pakolási probléma

Fogalmazzuk át most a színezési feladatot pakolásivá.

Egy adott G gráfról (14. ábra) el akarjuk dönteni, hogy k színnel színezhető-e a pontjai. Legyen G pontjainak száma n . Nyilván csak a $3 \leq k \leq n$ eset érdekes.

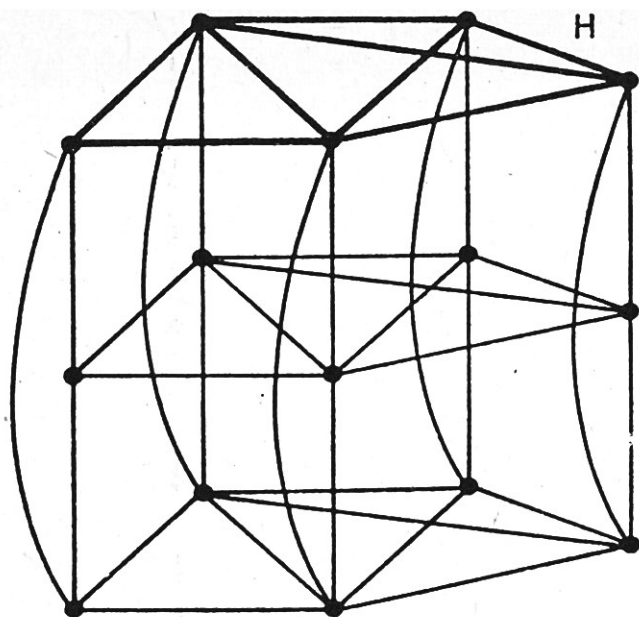


14. ábra

Vegyünk fel G -t k példányban: $G^{(1)}, \dots, G^{(k)}$, és kössük még össze a különböző példányok egymásnak megfelelő pontjait (15. ábra). Legyen az így nyert gráf H .

- **Állítás:** G akkor és csak akkor színezhető ki k színnel, ha H -ban van n független pont.

Tegyük fel először, hogy G kiszínezhető k színnel: nevezzük ezeket 1-es, 2-es $\dots$ színnek. Válasszuk ki $G^{(1)}$ -ből az 1-es, $G^{(2)}$ -ből a 2-es, $\dots$, $G^{(k)}$ -ből a k -as színű pontokat. Az így kiválasztott pon-



15. ábra

tok H -ban függetlenek. Továbbá G bármely x pontjának megfelelő pontok közül egyetlenegyét választottunk ki, hiszen x -nek egyetlenegy színe volt. Ezért a kiválasztott pontok száma n . Ezzel beláttuk, hogy ha G kiszínezhető k színnel, akkor H -ban van n független pont.

Megfordítva, tegyük fel, hogy H -ban van egy n elemű független ponthalmaz. Jelöljük ezt S -sel, és vizsgáljuk meg, hogyan helyezkedhet el S a H gráfban.

G minden egyes x pontjának H -ban k pont felel meg. Ezek páronként össze vannak kötve (így konstruáltuk H -t), ezért közülük legfeljebb egy tartozhat S -hez. De S -nek ugyanannyi pontja van, mint G -nek, így tehát G minden egyes pontjának kell is, hogy megfeleljen egy S -beli pont.

Színezzük ki G -t a következőképpen: G egy x pontját fessük ki az i színnel, ha az x -nek megfelelő H -beli pontok közül S éppen a $G^{(i)}$ -belit tartalmazza. Igen könnyű belátni, hogy az így definiált színezés jó, azaz az egymással összekötött G -beli pontok színe különböző.

Igazoltuk tehát, hogy ha H -ban van n független pont, akkor G kiszínezhető k színnel. Látjuk, hogy a gráfszínezési probléma visszavezethető a pakolási feladatra (a visszavezetés elég egyszerű is volt). Ezért a pakolási probléma is univerzális kereső feladat.

2.3.

Áttekintés

Foglaljuk össze a kereső feladatokra vonatkozó ismereteinket!

I. A normálformák kielégíthetőségének problémája *univerzális kereső feladat* (más szóval, *NP-teljes feladat*). Ezen azt értjük, hogy

minden kereső feladat *visszavezethető* rá. Pontosabban, minden kereső feladathoz szerkeszthető egy olyan normálforma (még hozzá a szerkesztést egy polinomiális algoritmus végzi), amely akkor és csak akkor elégíthető ki, ha az eredeti kereső feladatnak van megoldása. E normálformát kielégítő változóértékekből az eredeti kereső feladat egy megoldása „visszaszámolható”. Ez az állítás az algoritmus fogalmának pontosítása után teljes általánosságban igazolható. E könyvben speciális példákön mutattuk be a visszavezetést.

II. Vannak más univerzális kereső feladatok is, pl. a pakolási vagy gráfszínezési probléma. Egy feladatról azzal lehet bizonyítani, hogy univerzális, hogy — közvetve vagy közvetlenül — visszavezetjük rá a normálformák kielégíthetőségének problémáját. A visszavezetések sokfélék, minden egyes újabb feladat esetén újabb ötletet igényelnek.

Vannak olyan feladatok is, melyekről nem tudjuk, hogy univerzálisak-e. A legismertebb példa erre az *izomorfia-probléma*: Két gráfot izomorfnek nevezünk, ha „lényegében azonosak”, vagyis — pontosan — pontjaik úgy feleltethetők meg egymásnak, hogy két pont akkor és csakis akkor van összekötve az egyik gráfban, ha a megfelelő két pont össze van kötve a másikban. A probléma: döntjük el, hogy két adott gráf izomorf-e.

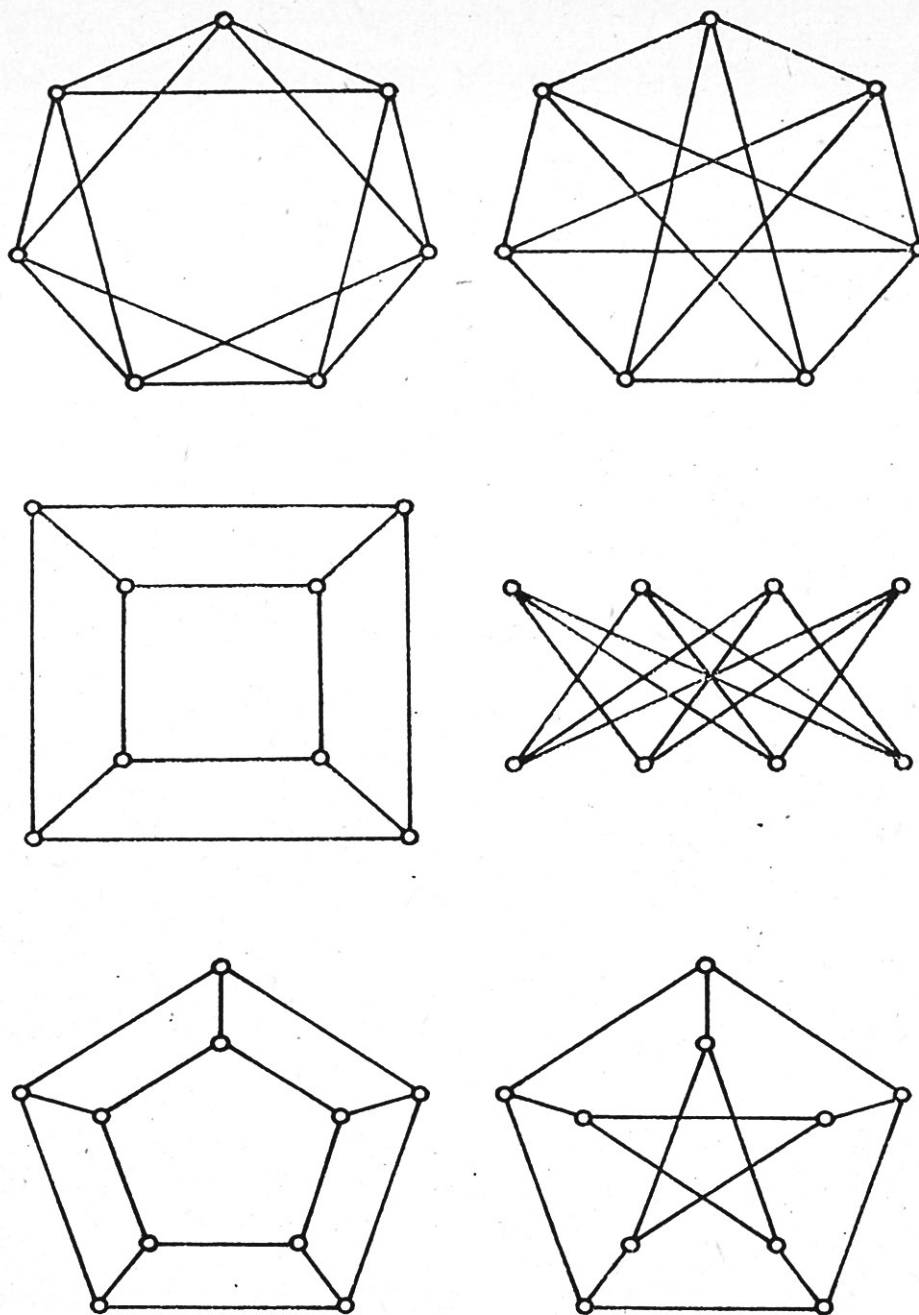
2.5. feladat: Az izomorfia-probléma nagyon könnyűnek tűnhet. Döntjük el a 16. ábrán látható gráfokról, melyek izomorfak közülük!

2.6. feladat: Mutassuk meg, hogy az izomorfia-probléma is kereső feladat.

Ha valaki eléggé sokat keresett izomorfia-tesztet (vagyis polinomiális algoritmust két gráf izomorf voltának eldöntésére), akkor egyre inkább megerősödik az a meggyőződés, hogy ez a feladat nem oldható meg polinomiális algoritmussal; hogy azonban univerzális lenne, azt eddig nem sikerült bebizonyítani.

III. Abból, hogy bármely kereső feladat visszavezethető bármely univerzálisra, következik, hogy ha az univerzális feladatok valamelyike megoldható polinomiális algoritmussal, akkor minden más kereső feladat is. Általánosan elfogadott (de nem bizonyított) *hipotézis*, hogy az univerzális kereső feladatok nem oldhatók meg polinomiális algoritmussal. Ilyenformán, ha egy kereső feladatról be lehet bizonyítani, hogy univerzális (vagyis a már ismert univerzális feladatok valamelyikét vissza lehet vezetni rá), akkor *úgy tekinthető*, hogy a polinomiális algoritmussal való megoldhatatlanságát is igazolták. (Ez nem azt jelenti, hogy a feladat megoldhatatlan, hanem azt, hogy a lassabb — általában exponenciális — algoritmusok között érdemes minél gyorsabbat, jobbat keresni.)

IV. Láttunk olyan kereső feladatokat (pl. a párosítási feladatot), melyek megoldhatók voltak polinomiális algoritmussal.



16. ábra

Ezek természetesen nem voltak univerzálisak. A megoldó algoritmusok viszonylag mély kombinatorikai meggondolásokon alapulnak és általános módszer az ilyen algoritmusok keresésére nincsen. Ezeknek a feladatoknak az osztályát P -vel szokás jelölni. Így $P \leq NP$, és a fentebb említett hipotézis úgy fogalmazható, hogy $P \neq NP$.

V Láttunk olyan kereső feladatokat, melyeknek volt *komplementáris feladatuk*, azaz egy olyan kereső feladat, mely azonos bemenő adatokhoz másvalamit keres, és az utóbbi akkor és csak akkor oldható meg, ha az eredeti nem. Az ilyen feladatokat szokás *jól karakterizáltnak* nevezni.

Kimutatható, hogy ha egy kereső feladatnak van polinomiális megoldó algoritmus, akkor van komplementáris feladata. Igaz-e

ez megfordítva? Nem tudjuk. Komplementáris feladatot találni gyakran könnyebb, mint polinomiális megoldó algoritmust, és így jó volna, ha az algoritmus kereséséhez a már megtalált komplementáris feladatot fel lehetne használni. Ilyen eredmények azonban nincsenek.

VI. A 4. fejezetben ismertetjük *Levin* algoritmusát, amely a kereső feladatokat bizonyos értelemben leggyorsabban oldja meg. Sajnos, nem tudjuk, mennyi ez a leggyorsabb: exponenciális vagy polinomiális.

A kereső feladatok vizsgálatának legfontosabb eredménye az, hogy általánosságban tudta fölvetni bizonyos feladatosztályok megoldó algoritmusainak a problémáját, és sok nemtriviális példa alapján érdekes és sokatmondó fogalmak megalkotását segítette elő (pl. univerzális kereső feladat vagy komplementáris feladat). Fontos, hogy érdekes és nehéz problémákat, sejtéseket is meg lehetett fogalmazni. Ez az elmélet ugyanakkor *nagyon egyszerűsít*, hiszen az algoritmusokat csak aszerint különbözteti meg, hogy a lefutásukhoz szükséges idő a feladat méretének (fix kitevőjű) hatványánál nem nagyobb-e. Két algoritmus, amelyek közül az egyik (n méretű feladaton) $n \log n$ lépésig, a másik n^{12} lépésig dolgozik, nagyon is különböző hatékonyságú. Másrészt egy $1,2^n$ lépésszámban dolgozó algoritmus tényleges problémák (tehát kis n) esetén sokkal használhatóbb lehet, mint egy $1000 n^{12}$ lépésszámú. Általánosságban mégis megállapítható: az, hogy mennyit értünk egy feladat háttéréből, matematikai finomságaiból, jól mérhető azzal, hogy tudunk-e rá polinomiális algoritmust adni.

Azok az eredmények, melyeket e fejezetben ismertettünk, csak az első lépéseket jelenthetik a feladatokat megoldó legrövidebb algoritmusok elméleti vizsgálatában, de megmutatják, hogy ilyen elmélet kidolgozható, és ezáltal az algoritmuselmélet egyik jövőbeli fejlődési irányát tűzik ki.

3. fejezet

Algebrai algoritmusok

Aki hagyományos középiskolai matematikaoktatásban részesült, olyan benyomást szerezhette, hogy a matematika algoritmusok gyűjteménye. Ezeknek az algoritmusoknak nagy részét már a régi görögök felfedezték, és a tanult eljárások az adott problémák megoldására a legjobbak, esetleg az egyedül ésszerűek. Ez a benyomás több szempontból is hibás. Először, mert mint a Bevezetésben hangsúlyoztuk, a matematika nem algoritmusok gyűjteménye, hanem annál jóval több. Másodszor: az iskolában tanult algoritmusok nem is olyan régiak. (Az algoritmus szó a IX. évszázadban élt *Mohamed ibn Musa al Chvarizmi* egyik könyvének latin címéből ered, amelyben az alpműveleteket és az Indiából származó tízes számrendszert ismerteti. Az algebra szó *al Chvarizmi* másik „*Hasib Aldzsabr Valmukabala*” című könyvéből származik.) A tizedes törtekkel való számolás szabályait például (különböző számrendszerekben) sokszor felfedezték a törté-

nelem folyamán, de alkalmazásuk csak a 17. században terjedt el. Az osztás elvégzése a 16. században még művészetnek számított és segédeszközök sokaságát vették hozzá igénybe. Érdekes módon a szorzás és osztás ma szokásos módja körülbelül egyidős a logaritmustáblával, mely ezt a két műveletet sok esetben feleslegessé teszi. Harmadszor, és ez lesz fejezetünk témája: az iskolában tanult algoritmusok *nem optimálisak*. Talán igaz, hogy legegyszerűbb megtanulni őket és nyomon követni működésüket, de az utóbbi időben a legősibb számolási feladatok megoldására a közismertnél sokkal gyorsabb eljárások is születtek. Az „algoritmus” szóról például legtöbbünknek az euklidészi algoritmus (két szám vagy polinom legnagyobb közös osztójának meghatározása) jut az eszébe. Ma már *Eukleidész* módszerét is egy gyorsabb váltotta fel.

Szeretnénk most megismertetni az olvasóval néhányat az új algoritmusokból, és azokból a vizsgálatokból is, amelyek az optimalitás igen nehéz kérdésével foglalkoznak.

Az új algoritmust nem azért ismertetjük, mert szerintünk máris ezt kellene alkalmazni, legalábbis a gépi számolásban. Az eljárás előnyei rendszerint csak nagy számokra, mátrixokra stb. érvényesülnek, viszont a leírása (program) bonyolultabb, a végrehajtás ellenőrzése még nem olyan kidolgozott, mint a régié.

Egy ténylegesen alkalmazott algoritmusnak *sok követelményt kell kielégítenie*. Mi rendszerint csak a szükséges műveletek számával foglalkozunk, és nem vizsgáljuk, mekkora a tárigény, hogyan halmozódnak az esetleges számolási hibák stb. Könnyen lehet, hogy a most legjobbnak tartott eljárások valamilyen szempontból rosszul vizsgáznak és így a gyakorlatban másoknak adják át a helyüket.

3.1.

Gyorsabban is lehet szorozni

Ebben a részben módszereket mutatunk be számok, mátrixok és polinomok szorzására.

3.1.1.

Nagy számok szorzása

Tegyük fel, hogy azt a nem túl gyakorlati feladatot kaptuk: számoljuk ki π értékét 40 értékes jegy pontossággal! Az iskolai módszer az lenne, hogy vesszük a számokat 40 értékes jegy pontossággal:

$$\pi = 3,14159\ 26535\ 89793\ 23846\ 26433\ 83279\ 50288\ 41972\ldots,$$

$$e = 2,71828\ 18284\ 59045\ 23536\ 02874\ 71352\ 66249\ 77572\ldots,$$

összeszorozzuk őket, majd a kapott 80 jegyű számot 40 jegyűre kerekítjük. Így nyilván $40^2 = 1600$ elemi műveletet kell végeznünk. (Megtakaríthatjuk a számolásnak körülbelül a felét, ha a részlet-szorzatokat csak a majdani 42-edik tizedesjegyig számoljuk.) Ha két egész számot, mondjuk u -t és v -t akarunk összeszorozni, ezek rendszerint valamilyen (pl. tízes) számrendszerben felírva állnak rendelkezésünkre:

$$u = \overline{u_{m-1} \dots u_1 u_0} = u_0 + 10u_1 + 100u_2 + \dots$$

$$v = \overline{v_{n-1} \dots v_1 v_0} = v_0 + 10v_1 + 100v_2 + \dots$$

Az eredményt

$$uv = w = \overline{w_{m+n-1} \dots w_1 w_0} = w_0 + 10w_1 + \dots$$

alakban kell megadni.

A hagyományos módszer ehhez mn darab elemi műveletet (tehát olyat, amelyet egyjegyű, azaz 10-nél kisebb számokkal végzünk) követel. Jobb eljárást először 1962-ben a szovjet *Karacuba* fedezett fel. Tegyük fel, hogy u is, v is $2n$ jegyű.

$$u = \overline{u_{2n-1} \dots u_0}, \quad v = \overline{v_{2n-1} \dots v_0}.$$

Ezeket a számokat így is írhatjuk:

$$u = 10^n U_1 + U_0, \quad v = 10^n V_1 + V_0,$$

ahol $U_1 = \overline{u_{2n-1} \dots u_n}$ az u „bal fele”, $U_0 = \overline{u_{n-1} \dots u_0}$ az u „jobb fele”, és ugyanígy $V_1 = \overline{v_{2n-1} \dots v_n}$, $V_0 = \overline{v_{n-1} \dots v_0}$.

Most a szorzatot így írhatjuk:

$$uv = 10^{2n} U_1 V_1 + 10^n (U_0 V_1 + U_1 V_0) + U_0 V_0.$$

Ennek a kifejezésnek a kiszámításában a négy $U_i V_j$ szorzat követeli a legtöbb műveletet. A döntő észrevétel az volt, hogy $U_0 V_1 + U_1 V_0$ kiszámításához elég egyetlen további szorzás:

$$U_0 V_1 + U_1 V_0 = (U_1 - U_0)(V_0 - V_1) + U_0 V_0 + U_1 V_1.$$

Végül is tehát uv így írható:

$$uv = (10^{2n} + 10^n) U_1 V_1 + 10^n (U_1 - U_0)(V_0 - V_1) + (10^n + 1) U_0 V_0.$$

Ezzel a formulával két $2n$ jegyű szám szorzását három (és nem négy) n jegyű számok közötti szorzásra redukáltuk. Könnyű utána-számolni, hogy a felmerülő összeadások együttesen még legfeljebb

$9n$ elemi műveletet tesznek ki. Ha $T(n)$ -nek jelöljük a két n jegyű szám szorzásához szükséges elemi műveletek számát, akkor tehát

$$T(2n) \leq 3T(n) + 9n. \quad (1)$$

Nyilván $T(1) = 1$. Alkalmazzuk az (1) képletet többször egymás után!

$$T(4n) \leq 3^2 T(n) + 3 \cdot 9n + 2 \cdot 9n = 3^2 T(n) + 9n(3^2 - 2^2),$$

$$T(8n) \leq 3^3 T(n) + 9n(3^3 - 3 \cdot 2^2) + 9 \cdot 2^2 n = 3^3 T(n) + 9(3^3 - 2^3)n.$$

Általában

$$T(2^k n) \leq 3^k T(n) + 9n(3^k - 2^k). \quad (2)$$

Legyen $n = 10$, $k = 2$! Ekkor

$$T(40) \leq 9T(10) + 450.$$

Mint tudjuk, $T(10) \leq 100$. Ezért $T(40) \leq 1350$. Ez persze még szerény megtakarítás az 1600-hoz képest. Eredeti feladatunkban, ahol a szorzatot nem 80, csak 40 jegy pontossággal kell ismerni, 8—900 művelet is elég volt. Minél nagyobbak a számok, annál nagyobb arányú a nyereség. Két n jegyű számot e módszerrel

$$27n^{\log_2 3} \approx 27n^{1,85}$$

lépésben szorozhatunk össze, az eredeti n^2 helyett.

3.1. feladat: Bizonyítsuk be a (2) egyenlőtlenséget és innen a

$$T(n) \leq 27n^{\log_2 3}$$

egyenlőtlenséget!

3.2. feladat: Alkalmazzuk *Karacuba* módszerét két 16-jegyű szám szorzására!

Karacuba módszerét azóta messzemenően általánosították. A ma ismeretes legjobb algoritmust a svájci *Schönhage* és *Strassen* adták meg. Ez a véges Fourier-transzformáltak felhasználásán alapul, és csak $cn \cdot \log n \cdot \log \log n$ számú műveletet igényel. Az általánosítás lényegét az a technika képezi, amelyet a polinomok szorzásáról szóló 3.1.3. pontban mutatunk be.

3.1.2.

Nagy mátrixok szorzása

Olyan nagy számok, melyek összeszorzásakor *Karacuba* módszere lényeges egyszerűsítést jelentene, talán nem is léteznek! Az ismert világmindenségben található elemi részecskék száma nem nagyobb, mint 10^{100} . Gyakran kell viszont nagy mátrixokkal műveleteket végezni. A 100×100 -as mátrixok szorzása, ilyen

méretű egyenletrendszerek megoldása mindennapos feladat, és csak a számítógépek kapacitása korlátozza a méretek növelését (például statikai számításokban vagy a gazdasági matematikában).

A mátrixszorzás bonyolultságának becsléséhez *kizárólag* az egyes mátrixelemekkel végzendő *algebrai műveleteket számoljuk össze*. Már az előző fejezetben sem számoltuk azokat a műveleteket, amelyeket az algoritmus megszervezése (számok ide-oda tologatása stb.) igényel. Ezek figyelembevétele az általunk vizsgált esetekben a műveletszám nagyságrendjén nem változtatna, de módosítaná a képet, mégpedig — különösen a mátrixszorzás esetén — a használt számítógéptől függően. (Ha nagy a mátrix, akkor esetleg el sem fér egyszerre a központi tárban.)

Tegyük fel, hogy össze akarunk szorozni két $n \times n$ -es mátrixot; legyenek ezek

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \text{ és } B = \begin{bmatrix} b_{11} & \dots & b_{1n} \\ \vdots & & \vdots \\ b_{n1} & \dots & b_{nn} \end{bmatrix}$$

A szorzat egy $n \times n$ -es mátrix, így írható:

$$C = \begin{bmatrix} c_{11} & \dots & c_{1n} \\ \vdots & & \vdots \\ c_{n1} & \dots & c_{nn} \end{bmatrix} \text{ ahol } c_{ij} = a_{i1}b_{1j} + \dots + a_{in}b_{nj} = \sum_{k=1}^n a_{ik}b_{kj}. \quad (3)$$

Ha a kijelölt műveleteket a felírt módon végezzük el, ez n^3 szorzást és $n^2(n-1)$ összeadást követel. *Strassen* észrevette, hogy ha A és B 2×2 -es mátrixok, akkor a szorzások számát 8-ról 7-re lehet csökkenteni, az összeadások számának növelése árán.

Képezzük ugyanis a következő 7 darab szorzatot:

$$\begin{aligned} u_0 &= (a_{11} + a_{22})(b_{11} + b_{22}), \\ u_1 &= (a_{21} + a_{22})b_{11}, \\ u_2 &= a_{11}(b_{12} - b_{22}), \\ u_3 &= a_{22}(-b_{11} + b_{21}), \\ u_4 &= (a_{11} + a_{12})b_{22}, \\ u_5 &= (-a_{11} + a_{21})(b_{11} + b_{12}), \\ u_6 &= (a_{12} - a_{22})(b_{21} + b_{22}). \end{aligned} \quad (4a)$$

Ellenőrizhető, hogy

$$\begin{aligned} c_{11} &= u_0 - u_3 - u_4 + u_6, \\ c_{21} &= u_1 + u_3, \\ c_{12} &= u_2 + u_4, \\ c_{22} &= u_0 + u_2 - u_1 + u_5. \end{aligned} \quad (4b)$$

A régi (3) képletek 8 szorzást és 4 összeadást követeltek, *Strassen* módszerével 7 szorzás és 18 összeadás vagy kivonás elegendő. Előnye még, hogy nem támaszkodik a szorzás kommutatív tulajdonságára, azaz az $ab=ba$ azonosságra, ezért mátrixokra is érvényes. Ha van egy $2n \times 2n$ -es A mátrixunk, akkor A -t így is írhatjuk:

$$A = \begin{bmatrix} A_{11} & : & A_{12} \\ \vdots & & \vdots \\ A_{21} & : & A_{22} \end{bmatrix},$$

ahol

$$A_{11} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}, \quad A_{12} = \begin{bmatrix} a_{1\ n+1} & \dots & a_{1\ 2n} \\ \vdots & & \vdots \\ a_{n\ n+1} & \dots & a_{n\ 2n} \end{bmatrix},$$

$$A_{21} = \begin{bmatrix} a_{n+1\ 1} & \dots & a_{n+1\ n} \\ \vdots & & \vdots \\ a_{2n\ 1} & \dots & a_{2n\ n} \end{bmatrix}, \quad A_{22} = \begin{bmatrix} a_{n+1\ n+1} & \dots & a_{n+1\ 2n} \\ \vdots & & \vdots \\ a_{2n\ n+1} & \dots & a_{2n\ 2n} \end{bmatrix}.$$

Ha B is egy $2n \times 2n$ -es mátrix, akkor könnyű látni, hogy a szorzat, $C=AB$, így írható:

$$C = \begin{bmatrix} C_{11} & : & C_{12} \\ \vdots & & \vdots \\ C_{21} & : & C_{22} \end{bmatrix}, \quad \text{ahol } C_{ij} = A_{11}B_{1j} + A_{12}B_{2j} \quad (i, j = 1, 2).$$

Legyen $S(n)$ az $n \times n$ -es mátrixok szorzásához szükséges összeadások, $M(n)$ pedig a szorzások száma. Ekkor *Strassen* képleteit mátrixokra alkalmazva azt kapjuk, hogy

$$M(2n) \leq 7M(n), \quad S(2n) \leq 18n^2 + 7S(n). \quad (5)$$

Nyilván $M(1)=1$, $S(1)=0$. Az (5) képletekből könnyen adódik, hogy

$$M(2^k) \leq 7^k, \quad S(2^k) \leq 6(7^k - 4^k). \quad (6)$$

Ha n tetszőleges, akkor $n \leq 2^k$, ahol $k = \lceil \log_2 n \rceil$. Így

$$M(n) \leq M(2^k) \leq 7^{\lceil \log_2 n \rceil} \leq 7 \cdot 7^{\log_2 n} = 7 \cdot n^{\log_2 7} \approx 7n^{2,81},$$

és hasonlóan

$$S(n) \leq 42n^{2,81}.$$

3.3. feladat: Bizonyítsuk be a (6) képleteket.

3.4. feladat: Ha a 2×2 -es mátrixokra a hagyományos szorzási módszerrel alkalmazzuk, akkor az

$$M(2n) \leq 8M(n), \quad S(2n) \leq 4n^2 + 8S(n)$$

képletekhez jutunk. Vezessük le innen az

$$M(2^k) \leq 8^k, \quad S(2^k) \leq 8^k - 4^k = e^{k \cdot \log 8} - e^{k \cdot \log 4} \quad (6a)$$

becsléseket!

Amint látjuk, azzal, hogy a 2×2 -es mátrixok szorzásakor 4 helyett 18 elemi összeadást és 1-gyel kevesebb elemi szorzást végzünk, a $2^k \times 2^k$ -as mátrixok szorzásakor az összeadások számát

$$S(2^k) \leq 6(7^k - 4^k) = 6(e^{k \cdot \log 7} - e^{k \cdot \log 4})$$

értékkel korlátozhatjuk.

A (6a)-beli $S(2k)$ megszorozódott 6-tal, viszont a kitevő $\frac{\log 7}{\log 8}$ - szor kisebb lett, és ha k nagy, akkor ez a döntő!

Karacuba szorzási módszeréhez hasonlóan, *Strassen* eredményét is sikerült tovább javítani. *Pan*, *Reif*, *Winograd* és *Strassen* eredményei alapján két, $n \times n$ -es mátrixot már $c \cdot n^{2.47}$ lépésben is össze lehet szorozni. Elképzelhető, hogy két mátrix összeszorozása csak $cn^{2+\varepsilon}$ műveletet igényel, ahol ε tetszés szerinti kicsinek választható.

3.1.3.

Polinomok szorzása

Az $x_1, \dots, x_k$ változók polinomjának nevezünk minden olyan kifejezést, amely ezekből a változókból és valós számokból a szorzás, összeadás és kivonás segítségével építhető fel. Pl. $F(x_1, x_2, x_3) = 1/4 + x_1^3 + 2x_2x_3$.

Egy egyváltozós, $(n-1)$ -ed fokú polinom, azaz egy

$$P(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

alakú kifejezés rendszerint az $a_0, \dots, a_{n-1}$ együttható-sorozattal van megadva. A

$$P(x) = a_0 + \dots + a_{n-1}x^{n-1}, \quad Q(x) = b_0 + \dots + b_{n-1}x^{n-1}$$

polinomokat össze is szorozhatjuk. Beszorzás és rendezés után újra egy

$$R(x) = P(x)Q(x) = c_0 + c_1x + \dots + c_{2n-2}x^{2n-2}$$

polinomot kapunk, amelynek c_i együtthatóit az alábbi képletek határozzák meg:

$$c_i = a_0b_i + a_1b_{i-1} + \dots + a_ib_0. \quad (7)$$

A $(c_0, \dots, c_{2n-2})$ sorozatot az $(a_0, \dots, a_{n-1})$ és $(b_0, \dots, b_{n-1})$ sorozatok *konvolúciójának* is szokták nevezni. Vegyük észre, hogy a számok szorzásának (3.1.1.) pontban tárgyalt feladata szoros kapcsolatban áll a polinomszorzás feladatával. Egy u szám b alapú számrendszerbeli felírása ugyanis

$$u = (u_{m-1} \dots u_0)_b = u_0 + u_1b + \dots + u_{m-1}b^{m-1}.$$

Ez nem más, mint a b szám olyan polinomjaként való felírás, ahol az együtthatók b -nél kisebbek. A szorzat kiszámítására persze a (7) képletek nem teljesen használhatók, mert b -nél nagyobb együtthatókat is előállíthatnak. Közelítőleg azonban jók lesznek, mert a szükséges korrekciók viszonylag kevés számú művelettel elvégezhetők. Ezért a 3.1.3. és 3.1.4. pontban bemutatott polinom-szorzási eljárások — megfelelő kiegészítésekkel — nagy számok gyorsabb szorzására is alkalmazhatók. A konvolúció művelete (a (7) képletekkel megadott együtthatók kiszámítása) önmagában is fontos gyakorlati feladat, például a statisztikában.

Ha a $c_0, c_1, \dots$ együtthatókat a felírt módon számoljuk ki, akkor n^2 darab szorzást kell végeznünk az a_i, b_j együtthatók között. A szovjet *Toom* észrevette, hogy a konvolúciót egyszerűbben végezhetjük el, ha figyelembe vesszük kapcsolatát a polinom-szorzással. Vegyünk $2n-1$ darab számot, pl. a

$$0, 1, \dots, 2n-2$$

számokat. Most ahelyett, hogy az $R(x) = P(x)Q(x)$ polinom együtthatóit számolnánk ki, először kiszámoljuk az $R(x)$ helyettesítési értékeit a $0, 1, \dots, 2n-2$ helyeken:

$$\begin{aligned} R(k) &= c_0 + c_1k + c_2k^2 + \dots = \\ &= (a_0 + a_1k + a_2k^2 + \dots)(b_0 + b_1k + b_2k^2 + \dots) = P(k)Q(k) \\ &(k=0, 1, \dots, 2n-2). \end{aligned}$$

Ehhez ki kell számolni $P(x)$ és $Q(x)$ helyettesítési értékeit a $0, 1, \dots, 2n-2$ helyeken, majd összeszorozni őket. Ezután hátravan még $R(x)$ együtthatóinak meghatározása az $R(0), R(1), \dots, R(2n-2)$ értékekből. Ez az interpolációs feladat a $c_0, c_1, \dots$ ismeretlenek kiszámítását jelenti a

$$\begin{aligned} c_0 &= R(0) \\ c_0 + c_1 + c_2 + \dots + c_{2n-2} &= R(1) \\ c_0 + 2c_1 + 2^2c_2 + \dots + 2^{2n-2}c_{2n-2} &= R(2) \\ &\vdots \\ c_0 + (2n-2)c_1 + \dots + (2n-2)^{2n-2}c_{2n-2} &= R(2n-2) \end{aligned} \tag{8}$$

lineáris egyenletrendszerből.

Van-e előnye ennek az eljárásnak? Ha az elvégzendő műveleteknek csak a számát tekintjük, akkor semmiképpen nincs, hiszen $P(0), P(1), \dots$ kiszámítása során n^2 , az egyenletrendszer megoldása során viszont $(2n-1)^3$ műveletet kell végezni. Tegyük azonban különbséget a *könnyebb* és a *nehezebb* műveletek között!

Könnyű műveleteknek tekintjük az összeadást és az olyan szorzást, ahol az egyik tényező ismert szám. Ezeket *lineáris műveleteknek* nevezzük.

$$P(2) = a_0 + a_1 \cdot 2 + a_2 \cdot 2^2 + \dots + a_{n-1} 2^{n-1}$$

értéke nyilván csupa lineáris művelettel kiszámítható. Hasonlóan

$$P(3), P(4), \dots, Q(0), Q(1), \dots$$

is kiszámolható lineáris műveletekkel.

Nehezebb művelet az olyan szorzás, melynél mindkét tényező paramétert tartalmazó kifejezés vagy olyan osztás, amelyben a nevező paramétert tartalmaz. Ezeket *nemlineáris műveleteknek* nevezzük.

A fenti eljárásban $P(0) \cdot Q(0)$, $P(1) \cdot Q(1)$, $\dots$ kiszámítása $2n-1$ darab nemlineáris műveletet kíván. Talán kevésbé nyilvánvaló, de a (8) egyenletrendszer megoldása csupa lineáris művelettel megvalósítható. Paraméterek ugyanis csak $R(0)$, $R(1)$, $\dots$, $R(2n-2)$ -ben vannak, ezeket pedig a megoldás során nem kell egymással se szorozni, se osztani. A nemlineáris műveletek száma tehát csak $2n-1$. A $c_0, c_1, \dots, c_{2n-2}$ együtthatókat definiáló eredeti képletekben n^2 darab nemlineáris művelet fordult elő. Ebből a szempontból tehát *Toom* itt leírt közvetett eljárása egyszerűbb.

3.5. feladat: Az interpolációs feladat egyenletrendszerrel való megoldása összesen $\textit{konstans} \cdot n^3$ műveletet kíván. Mutassuk meg, hogy a Newton-féle interpolációs módszerrel ezt a számot $\textit{konstans} \cdot n^2$ -re csökkenthetjük.

Kíváncsi, hogy lehetőleg azokból a műveletekből se kelljen sokat végezni, amelyeket könnyebbeknek nyilvánítottunk. $P(0)$, $P(1)$, $\dots$ kiszámítása a hagyományos módon kb. n^2 műveletet igényel, míg a (8) egyenletrendszer megoldásáé n^3 műveletet! (L. azonban a *Feladatot*.) Szerencsére a helyettesítési értékeket módunkban áll kedvezően megválasztani. Nem vagyunk a $0, 1, \dots, \dots, 2n-2$ számokhoz kötve, helyettük választhatunk akármilyen $\lambda_0, \lambda_1, \dots, \lambda_{2n-2}$ valós vagy komplex számokat is. Nagyon előnyös az a megoldás, amit a 3.1.4. pontban tárgyalunk. A $\lambda_0, \dots, \lambda_{2n-2}$ számokat bizonyos komplex egységgyököknek választjuk, és megmutatjuk, hogy így a konvolúció $c \cdot n \log n$ darab algebrai művelettel megkapható.

3.1.4.

A véges Fourier-transzformált

Tegyük fel, hogy adott az $a_0, \dots, a_{n-1}$ számsorozat, ill. az ehhez tartozó $P(x) = a_0 + a_1x + a_2x^2 + \dots$ polinom. Legyen r egy n -nél nem kisebb szám, és legyen

$$\varepsilon = \varepsilon_r = e^{\frac{2\pi i}{r}}$$

az első komplex r -edik egységgyök. Az

$$\varepsilon^0 = 1, \varepsilon, \varepsilon^2, \dots, \varepsilon^{r-1}$$

számok mind különbözők és $\varepsilon^r = 1$. Képezzük most a $P(1), P(\varepsilon), P(\varepsilon^2), \dots, P(\varepsilon^{r-1})$ értékeket:

$$\hat{a}_k = P(\varepsilon^k) = a_0 + a_1\varepsilon^k + a_2\varepsilon^{2k} + \dots + a_{n-1}\varepsilon^{(n-1)k} = F_k^{(r)}(a_0, \dots, a_{n-1})$$

$$(k = 0, \dots, r-1).$$

Az $(\hat{a}_0, \dots, \hat{a}_{r-1})$ komplex számsorozatot az $(a_0, \dots, a_{n-1})$ számsorozat véges Fourier-transzformáltjának nevezzük, mivel nem más, mint pl. a villamosmérnökök által is jól ismert és sokat használt Fourier-transzformált egy véges változata. E transzformációnak két tulajdonsága különösen fontos. Az egyik, az hogy a visszatranszformálás hasonló képletekkel történik: ha $r \geq n$, akkor

$$a_j = \frac{1}{r} (\hat{a}_0 + \hat{a}_1\varepsilon^{-j} + \hat{a}_2\varepsilon^{-2j} + \dots + \hat{a}_{r-1}^{-(r-1)j}) \quad (j = 0, \dots, n-1).$$

3.6. feladat: Bizonyítsuk be a fenti képletek helyességét!

A másik tulajdonság a definíciókból azonnal következik: két számsorozat konvolúciójának viszonylag bonyolult művelete a transzformált sorozatokon egy sokkal egyszerűbb műveletben jelenik meg. Ha $(c_0, \dots, c_{2n-2})$ az $(a_0, \dots, a_{n-1})$ és $(b_0, \dots, b_{n-1})$ sorozatok konvolúciója, és a Fourier-transzformálás során választott r nem kisebb, mint $2n-2$, akkor

$$\hat{c}_0 = \hat{a}_0\hat{b}_0, \quad \hat{c}_1 = \hat{a}_1\hat{b}_1, \quad \dots, \quad \hat{c}_{r-1} = \hat{a}_{r-1}\hat{b}_{r-1}.$$

A konvolúciót tehát így végezhetjük: transzformálás, r darab szorzás, visszatranszformálás.

Az eddigiekből az következik, hogy ha a 3.1.3. pontban ismertetett interpolációs módszerben a $\lambda_k = \varepsilon^k$ értékeket választjuk, akkor a nem számolt műveletek számát is kb. n^2 -re szoríthatjuk. (Feltételezzük, hogy a komplex egységgyökök a kívánt pontossággal rendelkezésünkre állnak. Nem változtatna azonban az el-

végzendő műveletek számának nagyságrendjén, ha azokat is magunknak kellene meghatározni). Ez az n^2 számú művelet első-sorban a *Fourier*-transzformáláshoz kell. Nem lehet-e gyorsabban transzformálni? Az amerikai *Cooley* és *Tukey* megadtak egy el-járást, amely a *Fourier*-transzformáltat $cr \log r$ számú művelettel kiszámolja. Most bemutatjuk ezt a — gyorsasága és egyszerűsége miatt nevezetes és széles körben alkalmazott — módszert.

Jelöljük $K(r)$ -rel az r -edik egységgyökökkel végzett *Fourier*-transzformáláshoz szükséges algebrai műveletek minimális számát. Legyen $a_0, \dots, a_{2r-1}$ egy $2r$ elemű sorozat. E sorozat transzformá-lásához az

$$\varepsilon_{2r} = e^{\frac{2\pi i}{2r}}$$

egységgyököket használjuk. Ezzel a jelöléssel

$$\hat{a}_k = a_0 + a_1 \varepsilon_{2r}^k + a_2 \varepsilon_{2r}^{2k} + \dots + a_{2r-1} \varepsilon_{2r}^{(2r-1)k}.$$

Bontsuk ezt az összeget két részre, különválasztva a páros és a páratlan indexű tagokat:

$$\begin{aligned} \hat{a}_k &= a_0 + a_2 \varepsilon_{2r}^{2k} + a_4 \varepsilon_{2r}^{4k} + \dots + a_1 \varepsilon_{2r}^k + a_3 \varepsilon_{2r}^{3k} + \dots = \\ &= a_0 + a_2 (\varepsilon_{2r}^2)^k + a_4 (\varepsilon_{2r}^2)^{2k} + \dots + \varepsilon_{2r}^k [a_1 + a_3 (\varepsilon_{2r}^2)^k + a_5 (\varepsilon_{2r}^2)^{2k} + \dots]. \end{aligned} \quad (9)$$

Vegyük most észre, hogy $\varepsilon_{2r}^2 = \varepsilon_r$. Ezért (9) a következő rekurzív formulát adja:

$$\begin{aligned} F_k^{(2r)}(a_0, \dots, a_{2r-1}) &= F_k^{(r)}(a_0, a_2, \dots, a_{2r-2}) + \\ &+ \varepsilon_{2r}^k F_k^{(r)}(a_1, a_3, \dots, a_{2r-1}). \end{aligned} \quad (10)$$

(10) alapján a szükséges műveletszám, $K(2r)$, így becsülhető: $\varepsilon_{2r}^2, \varepsilon_{2r}^3, \dots, \varepsilon_{2r}^{2r-1}$ számolása $2r-2$ darab szorzást követel. A (10) jobb oldalán álló két, r -ed rendű *Fourier*-transzformáláshoz definíció szerint $2K(r)$ művelet kell. (10) jobb oldalán még egy szorzás és egy összeadás van, ez, $2r$ -szer véve (hiszen minden k -ra el kell végezni), további $4r$ művelet. Ezért

$$K(2r) \leq 2K(r) + 6r. \quad (11)$$

Mivel $K(1)=0$, innen a (11) képletet m -szer egymásután alkalmazva,

$$K(2^m) \leq 3m \cdot 2^m.$$

Ha tehát r a 2-nek egy egész kitevőjű hatványa, akkor

$$K(r) \leq 3r \log_2 r.$$

Ha r nem 2^m alakú szám, akkor mindig megtehetjük, hogy az $a_0, \dots, a_{r-1}$ sorozatot nullákkal $a_0, \dots, a_{s-1}$ sorozattá egészítjük ki, ahol s egy 2-hatvány, majd a fenti módon egy s -ed rendű *Fourier*-transzformációt hajtunk végre.

3.2.

Osztás helyett szorzás

Nem nyilvánvaló, hogy az előző részben ismerttetett szorzási algoritmusok az osztást is könnyebbé teszik. Megmutatjuk ezért, hogyan lehet *az osztás feladatát is szorzásokkal megoldani*, számok, polinomok és mátrixok esetében.

Részletesen foglalkozunk az egyik legrégebbi nemtriviális algoritmussal, az *euklidészi algoritmussal* is. Ez az eljárás sok osztást tartalmaz, ezért hasznos, ha ezt a műveletet gyorsítani tudjuk. Figyelmet érdemel az a módszer is, amellyel ebben az algoritmusban további lényeges javítást értek el: elemezni kell, hogy az egyes lépések végrehajtásához milyen információra van valóban szükség, és az információt nem tartalmazó tagoktól meg kell szabadulni.

3.2.1.

Számok osztása

Ha u és v adott számok, akkor v/u -t általában csak *bizonyos pontosságig* számolhatjuk ki. Ha u, v egész számok, akkor esetleg kíváncsiak vagyunk a „hányados” és a „maradék” értékére is. A „hányados” nem más, mint $[v/u]$, azaz v/u egész része, ezért lényegében itt is v/u közelítő értékét keressük.

Az iskolában tanult osztási módszer nem tekinthető szigorú értelemben algoritmusnak, mert találgatás is kell hozzá. Ez a hiányosság azonban könnyen kiküszöbölhető, és így olyan osztási algoritmusunk lesz, amely ahhoz, hogy az n jegyű u és v számok hányadosát n értékes jegy pontossággal megadja, cn^2 elemi műveletet használ fel.

Most leírjuk *Newton* módszerét $1/u$ kiszámítására, ami a gyors szorzási módszerrel kombinálva a gyors osztást is lehetővé teszi. Tegyük fel az egyszerűség kedvéért, hogy a 2-es alapú számrendszerben dolgozunk. Azt is feltehetjük, hogy az u szám, amelynek a reciprokát keressük, 1 és 2 közé esik: $1 \leq u \leq 2$. Ha ez nem így volna, pl. $u = 0,001\ 111$ lenne, akkor — akárcsak a közönséges osztásnál — megtehetjük, hogy a „kettedesvesszőt” három hellyel jobbra toljuk, $1,111$ reciprokát számoljuk ki, majd az eredményben a „kettedesvesszőt” megint három hellyel jobbra toljuk. Ha

$1 \leq u \leq 2$, akkor nyilván $1/2 \geq 1/u \geq 1$. Vegyük ezért $1/u$ első megközelítésének $3/4$ -et: $x_1 = 3/4$. Ha már van egy m -edik x_m közelítésünk $1/u$ -ra, akkor a következő közelítést, x_{m+1} -et, így számoljuk:

$$x_{m+1} = x_m(2 - ux_m), \quad (1)$$

tehát pl. ha $x_1 = 3/4$, akkor

$$x_2 = \frac{3}{4} \left(2 - \frac{3}{4} u \right).$$

Győződjünk meg róla, hogy x_{m+1} valóban közelebb van $1/u$ -hoz, mint x_m volt! Tegyük fel, hogy $x_m = \frac{1}{u} + \varepsilon$, tehát x_m az $1/u$ -t ε hibával közelíti meg. Helyettesítsük ezt az (1) képletbe:

$$x_{m+1} = \left(\frac{1}{u} + \varepsilon \right) \left(2 - u \left(\frac{1}{u} + \varepsilon \right) \right) = \frac{1}{u} - u\varepsilon^2.$$

Ha tehát x_m hibája ε volt, akkor x_{m+1} hibája $u\varepsilon^2$, ami kisebb, mint $2\varepsilon^2$.

Ha x_m -et ismerjük, akkor $1/u$ -nak lényegében $\left\lceil \log \frac{1}{\varepsilon} \right\rceil$ darab értékes jegyét ismerjük. Jelöljük $\left\lceil \log \frac{1}{\varepsilon} \right\rceil$ -t L_m -mel! A fentiekből következik hogy

$$L_{m+1} \geq 2L_m - 1.$$

Mivel $L_1 \geq 2$, ezért

$$L_2 \geq 2^2 - 1, \quad L_3 \geq 2^3 - 2 - 1, \quad \dots,$$

$$L_m \geq 2^m - 2^{m-2} - 2^{m-3} - \dots - 1 = 2^m - 2^{m-1} + 1 > 2^{m-1}.$$

Ezért az m -edik közelítés értékes jegyeinek száma kb. 2^{m-1} . Ha az $1/u$ -t n értékes jegy pontossággal akarjuk tudni, akkor ezek szerint elég x_k -t kiszámolni, ahol $k = \lceil \log_2 n \rceil + 1$.

Minden iterációban, azaz minden újabb közelítés kiszámolásakor, 2^m -nél nem hosszabb számokkal két darab szorzást és kivonást végzünk. Az ehhez szükséges műveletek száma mint tudjuk, $c \cdot 2^m m \log_2 m$. Ezért az összes műveletek száma legfeljebb

$$\begin{aligned} c \sum_{m=1}^k 2^m m \log_2 m &< ck \log_2 k \sum_{m=1}^k 2^m < c \cdot 2^{k+1} k \log_2 k < \\ &< 8cn \log_2 n \log_2 \log_2 n, \end{aligned}$$

tehát nagyságrendileg nem több, mint amennyi a szorzáshoz kell.

3.2.2.

Hatványsorok és polinomok osztása

A polinomosztás, speciális algebrai jellege miatt, nem minden felsőbb matematikai tananyag része. Ez a művelet sok fontos, polinomokkal végzett algoritmus alapja, ezért foglalkozunk vele. Legyen

$$F(x) = a_m x^m + \dots + a_0, \quad G(x) = b_n x^n + \dots + b_0 \quad (a_m \neq 0, b_n \neq 0)$$

két polinom. Ha $F(x)$ -et elosztjuk $G(x)$ -szel, kapunk egy hányados-polinomot, $Q(x)$ -et, és egy maradékpolinomot, $R(x)$ -et, melyek az

$$F(x) = Q(x)G(x) + R(x) \quad (2)$$

egyenlőségnek tesznek eleget, és a maradékpolinom foka kisebb kell legyen, mint az osztónak, $G(x)$ -nek a foka. Ez a feltétel egyértelműen meghatározza a hányadost és a maradékot. Ha $m < n$, akkor $R(x) = F(x)$. Hagyjuk figyelmen kívül ezt a triviális esetet, és tegyük fel, hogy $m \geq n$.

Az osztás hagyományos algoritmusát egy egyszerű példán mutatjuk be. Legyen

$$F(x) = x^3 + 1, \quad G(x) = 2x^2 - x.$$

Először elosztjuk $F(x)$ legmagasabb fokú tagját $G(x)$ legmagasabb fokú tagjával: az eredmény $\frac{1}{2}x$, ez lesz $Q(x)$ legmagasabb fokú tagja. Ezután beszorozzuk $G(x)$ -et $\frac{1}{2}x$ -szel, és az eredményt levonjuk $F(x)$ -ből; így kapjuk az első maradékot:

$$R_1(x) = F(x) - \frac{1}{2}xG(x) = \frac{1}{2}x^2 + 1.$$

$R_1(x)$ foka legalább eggyel kisebb, mint $F(x)$ foka. Most $R_1(x)$ legmagasabb fokú tagját osztjuk el $G(x)$ legmagasabb fokú tagjával: az eredmény $\frac{1}{4}$. Ez lesz $Q(x)$ második tagja. A második maradék

$$R_2(x) = R_1(x) - \frac{1}{4}G(x) = \frac{1}{4}x + 1.$$

Mivel $R_2(x)$ foka kisebb, mint $G(x)$ -é, ez a végső maradék is:

$$Q(x) = \frac{1}{2}x + \frac{1}{4}, \quad R(x) = \frac{1}{4}x + 1,$$

és valóban teljesül a (2) egyenlőség:

$$x^3 + 1 = \left(\frac{1}{2}x + \frac{1}{4}\right)(2x^2 - x) + \left(\frac{1}{4}x + 1\right).$$

Ha a hányadosban x negatív kitevőjű hatványait is megengednénk, akkor az osztást formálisan a végtelenségig folytathatnánk.

3.7. feladat: Az osztás folytatásával mutassuk meg, hogy

$$\frac{x^3 + 1}{2x^2 - x} = \frac{1}{2}x + \frac{1}{4} + \frac{1}{8x} + \frac{9}{4} \left(\frac{1}{(2x)^2} + \frac{1}{(2x)^3} + \frac{1}{(2x)^4} + \dots \right).$$

3.8. feladat: Mutassuk meg, hogy általában, ha

$$F(x) = a_m x^m + \dots + a_0, \quad G(x) = b_n x^n + \dots + b_0 \quad (a_m, b_n \neq 0)$$

két polinom, akkor polinomosztással kiszámolható a

$$c_{m-n}x^{m-n} + \dots + c_1x + c_0 + c_{-1}x^{-1} + c_{-2}x^{-2} + \dots = \frac{F(x)}{G(x)}$$

„negatív kitevőjű hatványsor”, és

$$Q(x) = c_{m-n}x^{m-n} + \dots + c_0.$$

Belátható, hogy ha az osztási eljárást egy tetszőleges m -ed fokú $F(x)$ és egy tetszőleges n -ed fokú $G(x)$ polinomra alkalmazzuk, akkor végrehajtása során az együtthatókon $cn(m-n)$ algebrai műveletet kell végezni. A svájci *Sieveking* adott ennél lényegesen gyorsabb eljárást. Ennek ismertetése előtt a polinomok általánosításával, a formális hatványsorokkal foglalkozunk. Egy

$$S(x) = c_0 + c_1x + c_2x^2 + \dots$$

hatványsor végtelen polinomnak is felfogható. Számunkra jelenleg nem érdekes, konvergens-e a sor x valamilyen adott értéke esetén. A hatványsorok összeadása és szorzása formálisan ugyanúgy történik, mint a polinomoké.

Gyakorlatilag persze sohasem végtelen hatványsorokkal dolgozunk, hanem megfelelő véges közelítéseikkel. Legyen

$$S_k(x) = c_0 + c_1x + \dots + c_{k-1}x^{k-1}.$$

Ha az $S(x)$ és $T(x)$ hatványsorok az x^{k-1} együtthatójáig meg egyeznek, azaz $S_k(x) = T_k(x)$, akkor ezt így jelöljük:

$$S(x) \equiv T(x) \pmod{x^k}$$

[olvasd: $S(x)$ kongruens $T(x)$ -szel modulo x^k]. Ezzel a jelöléssel a moduláris algoritmusokról szóló részben is találkozunk. Jelen-

tése: $S(x) - T(x)$ „osztható” x^k -val, $S(x)$ és $T(x)$ „ugyanazt a maradékot adja” x^k -val osztva. Ha

$$S(x) = c_0 + c_1x + \dots, \quad T(x) = d_0 + d_1x + \dots,$$

akkor keressük azokat az $e_0, e_1, \dots$ együtthatókat, amelyekre a

$$(c_0 + c_1x + \dots)(e_0 + e_1x + \dots) = d_0 + d_1x + \dots$$

egyenlet teljesül. A jobb és bal oldalon x azonos hatványaihoz tartozó együtthatók összehasonlításából látszik, hogy ez az egyenlet nem oldható meg akkor, ha pl. $c_0 = 0, d_0 \neq 0$. Sőt, a megoldhatóság szükséges feltétele az, hogy ha $S(x)$ első k darab együtthatója 0, akkor $T(x)$ első k darab együtthatója is 0 legyen. Ha $S(x)$ első nem 0 együtthatója c_k , és a fenti szükséges feltétel teljesül, akkor mindkét oldalt eloszthatjuk x^k -val. Ezért mindig feltételezhetjük, hogy $c_0 \neq 0$.

Ha $c_0 \neq 0$, akkor $S(x)$ -nek mindig van reciproka, azaz olyan $V(x) = f_0 + f_1x + \dots$ hatványsor, amelyre $S(x)V(x) = 1$. (Ennek segítségével

$$\frac{T(x)}{S(x)} = T(x)V(x),$$

tehát az $S(x)$ -szel való osztás mindig elvégezhető.)

Az $f_0, f_1, f_2, \dots$ együtthatókat könnyű a

$$(c_0 + c_1x + c_2x^2 + \dots)(f_0 + f_1x + f_2x^2 + \dots) = 1$$

egyenletből együttható-összehasonlítással megkapni:

$$\begin{array}{ll} c_0f_0 = 1 & \text{azaz} \quad f_0 = 1/c_0 \\ c_0f_1 + c_1f_0 = 0 & \text{azaz} \quad f_1 = -c_1f_0/c_0 \\ c_0f_2 + c_1f_1 + c_2f_0 = 0 & \text{azaz} \quad f_2 = -(c_1f_1 + c_2f_0)/c_0 \\ \text{stb.} \end{array}$$

A reciproka hatványsor i -edik együtthatója tehát az eredeti hatványsor első i darab együtthatója segítségével határozható meg:

$$V_k(x) \equiv 1/S_k(x) \pmod{x^k}.$$

Könnyű utánaszámolni, hogy ez a módszer $V_k(x)$ kiszámolásához $\text{konstans} \cdot k^2$ műveletet kíván. Most bemutatjuk Sieveking módszerét, amely ezt a számot $\text{konstans} \cdot k \cdot \log k$ -ra, a nemlineáris műveletek számát pedig $\text{konstans} \cdot k$ -ra csökkenti.

Mint az összes eddigi gyors módszerben, a $2k$ hosszú sor, $V_{2k}(x) = f_0 + f_1x + \dots + f_{2k-1}x^{2k-1}$ kiszámítását fogjuk k hosszúságú sorokkal végzett műveletekre visszavezetni.

Fennáll

$$(f_0 + f_1x + \dots + f_{2k-1}x^{2k-1})(c_0 + c_1x + \dots + c_{2k-1}x^{2k-1}) \equiv 1 \pmod{x^{2k}}.$$

Átalakítva:

$$V_k(x)S_{2k}(x) + x^k(f_k + f_{k+1}x + \dots + f_{2k-2}x^{k-1})S_k(x) + x^{2k}(f_k + \dots + f_{2k-1}x^{k-1})(c_k + \dots + c_{2k-1}x^{k-1}) \equiv 1 \pmod{x^{2k}}. \quad (3)$$

A bal oldal utolsó tagja x^{2k} -val osztható, ezért a kongruencia akkor is fennáll, ha ezt a tagot elhagyjuk. Vezessük be a következő jelölést:

$$V_k(x)S_{2k}(x) = g_0 + g_1x + \dots$$

Mivel $V_k(x)$ az $S(x)$ k -pontosságú reciproka, így

$$g_0 = 1, \quad g_1 = g_2 = \dots = g_{k-1} = 0.$$

E jelölés segítségével (3) így írható:

$$x^k(g_k + g_{k+1}x + \dots + g_{2k-1}x^{k-1}) + x^k(f_k + \dots + f_{2k-1}x^{k-1})S_k(x) \equiv 0 \pmod{x^{2k}}.$$

Ez nyilván azt jelenti, hogy

$$(f_k + \dots + f_{2k-1}x^{k-1})S_k(x) \equiv -(g_k + \dots + g_{2k-1}x^{k-1}) \pmod{x^k}.$$

Szorozzuk meg mindkét oldalt $V_k(x)$ -szel:

$$f_k + \dots + f_{2k-1}x^{k-1} \equiv -(g_k + \dots + g_{2k-1}x^{k-1})V_k(x) \pmod{x^k}. \quad (4)$$

Ez a képlet használható tehát $f_k, \dots, f_{2k-1}$ meghatározására. Jelölje $M(k)$ a két k -ad fokú polinom összeszorzásához szükséges műveletek számát, $L(k)$ a $V_k(x)$ kiszámításához szükséges műveletek számát. Ha a (4) rekurzív képlet alapján számolunk, akkor azt kapjuk, hogy

$$L(2k) \leq \underbrace{L(k)}_{\text{kiszámolása}} + \underbrace{M(2k)}_{\text{kiszámolása}} + \underbrace{M(k)}_{\text{a (4) képletbeli szorzás}} \quad (5)$$

Az előző részben láttuk, hogy két k -ad fokú polinom összeszorzása c_1k nemlineáris műveletet igényel, azaz

$$M_{\text{nemlin}}(k) \leq c_1k.$$

Az összes műveletek számára becslésünk a következő volt:

$$M_{\text{össz}}(k) \leq c_2k \log k$$

(c_1 és c_2 konstansok). Innen és (5)-ből adódik, hogy

$$L_{\text{nemlin}}(k) \leq c_3k, \quad L_{\text{össz}}(k) \leq c_4k \log k, \quad (6)$$

ahol c_3 és c_4 ugyancsak konstansok, tehát az osztás nem követel nagyságrendileg több műveletet, mint a szorzás.

Feladat: Bizonyítsuk be a (6) képleteket.

Alkalmazzuk most az itt kapott hatványsorozási eljárást a polinomosztás feladatára! Adva vannak tehát az

$$F(x) = a_m x^m + \dots + a_0, \quad G(x) = b_n x^n + \dots + b_0$$

polinomok, célunk a (2) képletet kielégítő $Q(x)$ hányados kiszámítása.

E pont feladataiból láttuk, hogy a polinomosztást hatványsorozásnak is felfoghatjuk. Most azonban nem x , hanem $\frac{1}{x}$ hatványai szerinti végtelen sorokkal van dolgunk. A közönséges hatványsorokra kapott eredmény felhasználásához vezessük be az $y = \frac{1}{x}$,

$$S(y) = a_m + a_{m-1}y + \dots + a_0 y^m,$$

$$T(y) = b_n + \dots + b_0 y^n$$

jelölést. Osszuk el egymással az $S(y)$, $T(y)$ hatványsorokat:

$$\frac{S(y)}{T(y)} = e_0 + e_1 y + e_2 y^2 + \dots$$

Ekkor

$$\begin{aligned} \frac{F(x)}{G(x)} &= \frac{x^m S(x^{-1})}{x^n T(x^{-1})} = x^{m-n} (e_0 + e_1 x^{-1} + \dots) = \\ &= e_0 x^{m-n} + \dots + e_{m-n} + e_{m-n+1} \frac{1}{x} + e_{m-n+2} \frac{1}{x^2} + \dots, \end{aligned}$$

tehát a hányados

$$Q(x) = e_0 x^{m-n} + \dots + e_{m-n}.$$

Az $F(x)$ és $G(x)$ polinomok osztásának feladatát így visszavezettük az $S(y)$ és $T(y)$ hatványsorok osztására. A szükséges műveletek számára ugyanaz a becslés adható, mint az előbb:

$$L_{\text{nemlin}}(m) \leq c_3 m, \quad L_{\text{össz}}(m) \leq c_4 m \log_2 m.$$

3.2.3.

Az euklidészi algoritmus

Két egész szám, a és b legnagyobb közös osztójának azt a legnagyobb d egész számot nevezzük, amely mindkettőnek osztója. Ezt a számot hagyományosan így jelöljük:

$$d = (a, b).$$

Ismeretes a legnagyobb közös osztó következő tulajdonsága is: ha a c szám közös osztója a -nak és b -nek is, akkor c osztója d -nek is. Előfordul, hogy az a és b számoknak nincs 1-nél nagyobb közös osztója, ekkor tehát $(a, b) = 1$. Ilyen esetben azt mondjuk, hogy a és b relatív prímek.

Mindezek a fogalmak (oszthatóság, legnagyobb közös osztó stb.) értelmezhetők polinomokra is. Itt legnagyobb közös osztón a legmagasabb fokszámú közös osztót értjük. Az is fennáll, hogy az $F(x)$ és $G(x)$ polinomok legnagyobb közös osztója, $D(x)$, osztható $F(x)$ és $G(x)$ minden más közös osztójával. Megjegyezzük, hogy $(F(x), G(x))$ csak konstans szorzó erejéig van egyértelműen meghatározva.

Gyakorlati számítási feladatokban szükségünk lehet a legnagyobb közös osztó meghatározására.

Mindenekelőtt, ha törtekkel dolgozunk (akár számokról, akár polinomokról van szó), akkor többnyire a legegyszerűbb alakban szeretnénk őket tárolni. Az

$$\frac{a}{b}$$

tört egyszerűsítése éppen azt jelenti, hogy a -t is, b -t is elosztjuk a és b legnagyobb közös osztójával:

$$\frac{a}{b} = \frac{a'}{b'} \text{ ahol } a' = \frac{a}{(a, b)}, \quad b' = \frac{b}{(a, b)}.$$

Emlékezzünk vissza arra is, hogyan tanultuk két tört, a/b és c/d összeadását: az összeget számolhatjuk az

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$$

képlet alapján is, de ha b és d egész számok (vagy polinomok), akkor közös nevezőnek elég lehet egy bd -nél kisebb szám is, ami

közös többszöröse b -nek és d -nek. Lehet természetesen a legkisebb közös többszöröst is választani, ezt hagyományosan

$$[b, d]$$

-vel jelöljük. Ismeretes, hogy

$$[b, d] = \frac{bd}{(b, d)},$$

ezért a két tört összeadásának képlete így alakul:

$$\frac{a}{b} + \frac{c}{d} = \frac{ad' + b'c}{[b, d]}, \quad \text{ahol } b' = \frac{b}{(b, d)}, \quad d' = \frac{d}{(b, d)}.$$

A legnagyobb közös osztó kiszámítása fontos lehet akkor is, ha csak annak megállapítása érdekel bennünket, hogy két szám (vagy polinom) relatív prím-e? Fejezetünknek a moduláris algoritmusokról szóló részében példákat látunk majd olyan esetekre, amikor sok, páronként relatív prím $m_1, \dots, m_k$ számra van szükség. A $(\text{mod } p)$ vett osztás, ez a sok moduláris eljárásban szereplő alapl művelet, szintén az euklidészi algoritmus segítségével végezhető el.

Mielőtt egy további, kevésbé triviális példát mutatnánk alkalmazására, lássuk *Eukleidész* algoritmusát!

Célunk két természetes szám, mondjuk a_0 és a_1 legnagyobb közös osztójának meghatározása.

Osszuk el az a_0 számot a_1 -gyel maradékosan! Legyen a hányados q_1 , a maradék pedig a_2 , egy a_1 -nél kisebb természetes szám:

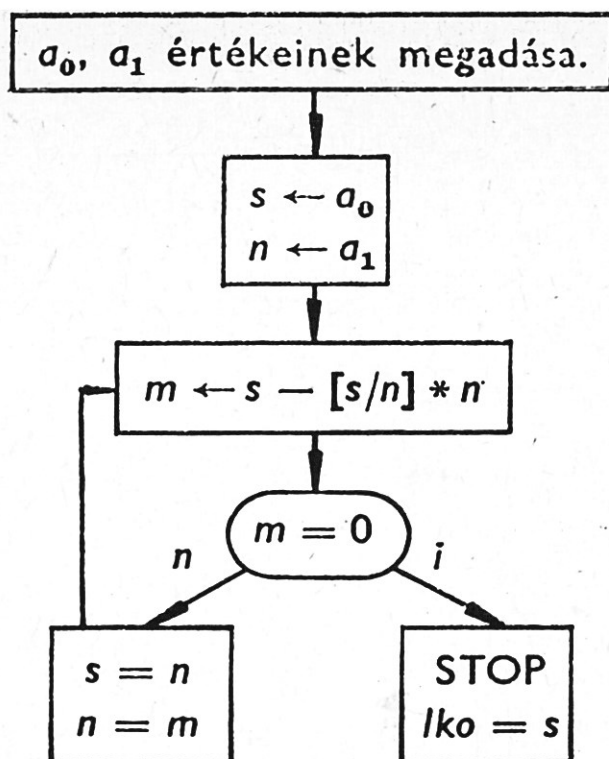
$$a_0 = q_1 a_1 + a_2 \quad (a_2 < a_1).$$

(Ez és minden, ami ezután jön, polinomokra is érvényes: ilyenkor „ $a_2 < a_1$ ” helyett „ $F_2(x)$ foka kisebb, mint $F_1(x)$ foka” értendő.) Hajtsuk most végre egymás után a következő maradékos osztásokat:

$$\begin{aligned} a_0 &= q_1 a_1 + a_2, & (a_2 < a_1), \\ a_1 &= q_2 a_2 + a_3, & (a_3 < a_2), \\ &\vdots \\ a_{k-1} &= q_k a_k + a_{k+1}, & (a_{k+1} < a_k). \end{aligned}$$

Ha $a_{k+1} = 0$, akkor megállunk, ellenkező esetben folytatjuk az eljárást.

3.9. feladat: Lássuk be teljes indukcióval, hogy ha a_{k+1} az első nulla maradék, akkor a_k lesz a_0 és a_1 legnagyobb közös osztója (mint ahogy az $a_0, \dots, a_k$ sorozat bármely két, egymás után következő elemének legnagyobb közös osztója is).



A

$(q_1, q_2, \dots, q_k, a_k)$

sorozatot az a_0 és a_1 számok *hányadosvektorának* nevezzük.

Az euklidészi algoritmus igen hatékonyan dolgozik: ha a_0 és a_1 (bináris felírásban) n -nél nem több jegyű szám (ill. legfeljebb n -ed fokú polinom), akkor kb. n lépésben (rendszerint azonban sokkal gyorsabban) véget ér. Érdekes megjegyezni, hogy máig sem áll rendelkezésünkre elég gyors algoritmus annak eldöntésére, hogy egy adott b szám prím-e vagy sem (adott polinom irreducibilis-e vagy sem), de az euklidészi algoritmussal gyorsan el tudjuk dönteni, hogy a_0 és a_1 számok relatív prímek-e.

Az euklidészi algoritmus által szerzett információból sokszor nemcsak a legnagyobb közös osztó értéke az, amit hasznosítani tudunk, hanem a teljes $(q_1, \dots, q_k, a_k)$ hányadosvektor. A hányadosvektorból az $a_2, \dots, a_k$ számok bármelyike már könnyen meghatározható.

Definiáljuk ugyanis a következő mátrixokat:

$$M_i = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix}. \quad (7)$$

Az M_i mátrix segítségével lehet az $a_0, a_1, \dots, a_k$ sorozatban „eggyel továbblépni”:

$$\begin{bmatrix} a_i \\ a_{i+1} \end{bmatrix} = M_i \cdot \begin{bmatrix} a_{i-1} \\ a_i \end{bmatrix}.$$

Ezért

$$\begin{bmatrix} a_i \\ a_{i+1} \end{bmatrix} = M_1 M_2 \dots M_i \cdot \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}.$$

Az $M_1 M_2 \dots M_i$ mátrixszorzat a $q_1, \dots, q_i$ sorozat meghatározásával párhuzamosan és éppoly gyorsan kiszámítható. Legyen

$$M_1 M_2 \dots M_i = \begin{bmatrix} u_{11}^{(i)} & u_{12}^{(i)} \\ u_{21}^{(i)} & u_{22}^{(i)} \end{bmatrix}.$$

Ekkor speciálisan

$$a_k = u_{11}^{(k)} a_0 + u_{12}^{(k)} a_1.$$

Tehát az euklidészi algoritmus mellékeredményként olyan v_0, v_1 számokat is létrehoz, melyek segítségével a_0 és a_1 legnagyobb közös osztója egyszerű módon fejezhető ki:

$$(a_0, a_1) = v_0 a_0 + v_1 a_1. \quad (8)$$

A v_0, v_1 számokra a moduláris algoritmusok tárgyalásakor még szükségünk lesz.

Lássuk most a hányadosvektor egy fontos alkalmazását!

Tegyük fel, hogy az $F(x)$ polinomról el kell dönteni, van-e gyöke az (u, v) intervallumban, ahol u és v adott valós számok, és $u < v$. A múlt században *Sturm* jött rá arra, hogy az euklidészi algoritmus segítségével még az u és v között található gyökök számát is meg lehet határozni.

Módszere csak a különböző gyökök számát adja meg, vagyis azt nem tudja eldönteni, hogy a gyökök hányszorosak. (Az $F(x)$ polinomnak a w szám m -szeres gyöke, ha $F(x)$ osztható $(x-w)^m$ -nel.)

A gyökök számának meghatározásához hasznunkra van a következő algebrai tétel:

• **Tétel:** Képezzük az

$$F(x) = u_0 + u_1 x + u_2 x^2 + \dots + u_n x^n$$

polinom deriváltját:

$$F'(x) = u_1 + 2u_2 x + \dots + nu_n x^{n-1}.$$

Legyen $F(x)$ és $F'(x)$ legnagyobb közös osztója $D(x)$ és osszuk el ezzel $F(x)$ -et:

$$\tilde{F}(x) = F(x)/D(x).$$

Az így kapott $\tilde{F}(x)$ gyökei megegyeznek $F(x)$ gyökeivel, de $\tilde{F}(x)$ minden gyöke egyszeres.

Mivel alacsonyabb fokú polinomokkal könnyebb dolgozni, rendszerint érdemes elvégezni azt a redukciót, amit a fenti tétel lehetővé tesz, és $F(x)$ helyett $\tilde{F}(x)$ -szel dolgozni. Ezért a továbbiakban *feltesszük, hogy $F(x)$ minden gyöke egyszeres* (azaz $F(x) = \tilde{F}(x)$). Ez más szóval azt jelenti, hogy $(F(x), F'(x)) = 1$.

Alkalmazzuk az euklidészi algoritmust az

$$F_0(x) = F(x), \quad F_1(x) = F'(x)$$

polinomokra:

$$\begin{aligned} F_0(x) &= F_1(x)Q_1(x) + F_2(x), \\ F_1(x) &= F_2(x)Q_2(x) + F_3(x), \\ &\vdots \\ F_{k-1}(x) &= F_k(x)Q_k(x). \end{aligned}$$

Itt $F_k(x) = (F_0(x), F_1(x))$, az $F(x)$ és $F'(x)$ legnagyobb közös osztója. Feltetésünk szerint $F(x)$ és $F'(x)$ relatív prímek, tehát $F_k(x)$ egy nullától különböző állandó lesz. Vegyük most a $Q_1(x) \dots Q_k(x)$ sorozat helyett a $\bar{Q}_1(x) \dots \bar{Q}_k(x)$ sorozatot:

$$\bar{Q}_1 = Q_1, \bar{Q}_2 = -Q_2, \bar{Q}_3 = Q_3, \bar{Q}_4 = -Q_4, \dots$$

és az $F_0(x), F_1(x), \dots, F_k(x)$ sorozat helyett az $\bar{F}_0(x), \dots, \bar{F}_k(x)$ sorozatot, amely abban tér el az előzőtől, hogy minden harmadik és negyedik elem előjele az ellenkezőjére változik:

$$\begin{aligned} \bar{F}_0 &= F_0, \bar{F}_1 = F_1, \bar{F}_2 = -F_2, \bar{F}_3 = -F_3, \bar{F}_4 = F_4, \bar{F}_5 = F_5, \\ \bar{F}_6 &= -F_6, \dots \end{aligned}$$

3.10. feladat: Lássuk be, hogy az új sorozatok az eredeti euklidészi algoritmushoz igen hasonló képleteknek tesznek eleget:

$$\begin{aligned} \bar{F}_0(x) &= \bar{F}_1(x)\bar{Q}_1(x) - \bar{F}_2(x), \\ \bar{F}_1(x) &= \bar{F}_2(x)\bar{Q}_2(x) - \bar{F}_3(x), \\ \bar{F}_2(x) &= \bar{F}_3(x)\bar{Q}_3(x) - \bar{F}_4(x). \end{aligned} \tag{8}$$

Mire jó az $\bar{F}_0(x), \dots, \bar{F}_k(x)$ sorozat ismerete? Emlékezzünk vissza eredeti feladatunkra: el akarjuk dönteni, van-e az $F(x)$ polinomnak gyöke az u és v pontok közötti intervallumon. *Sturm* megmutatta, hogy ez kiolvasható az $\bar{F}_0(u), \dots, \bar{F}_k(u), \bar{F}_0(v), \dots, \bar{F}_k(v)$ számok előjeleiből, feltéve, hogy e számok egyike sem nulla. (Ha az igen kevéssé valószínű eset állna elő, hogy e számok valamelyike nulla, akkor meg kell kissé változtatnunk a vizsgált (u, v) intervallumot.) Jelöljük $w_F(u)$ -val az $\bar{F}_0(u), \dots, \bar{F}_k(u)$ sorozatban található előjelváltások számát! Az alábbi sorozatban például a vonással jelölt helyeken vannak előjelváltások, ezért számuk három:

$$3 \quad 2 \quad | \quad -8 \quad -5 \quad -11 \quad | \quad 116 \quad | \quad -1$$

- **Tétel (Sturm):** Ha $\bar{F}_0(x), \dots, \bar{F}_k(x)$ egyike sem tűnik el az u és v helyeken, és $F(x)$ minden gyöke egyszeres az (u, v) intervallumban, akkor $F(x)$ -nek az (u, v) intervallumban pontosan $w_F(u) - w_F(v)$ darab gyöke van.

Példa: Hány gyöke van az $F(x) = x^3 - 3x + 1$ függvénynek 0 és $3/2$ között?

$$F'(x) = 3x^2 - 3, \quad \bar{F}_0(x) = F(x), \quad \bar{F}_1(x) = F'(x).$$

Innen a (8) képletek alapján számolunk tovább:

$$\bar{F}_0(x) = \frac{1}{3}x \bar{F}_1(x) - (2x - 1),$$

azaz

$$\bar{F}_2(x) = 2x - 1.$$

$$\bar{F}_1(x) = \frac{3}{2}\bar{F}_2(x) - 1/2, \quad \text{azaz} \quad \bar{F}_3(x) = 1/2.$$

A 0-hoz tartozó előjelsorozat $\bar{F}_0(0), \bar{F}_1(0), \bar{F}_2(0), \bar{F}_3(0)$ előjeleiből áll: ezek

$$+ \quad | \quad - \quad - \quad | \quad +$$

(2 előjelváltás).

A $3/2$ -hez tartozó előjelsorozat

$$- \quad | \quad + \quad + \quad +$$

(1 előjelváltás).

Ezért a vizsgált intervallumon a gyökök száma $2 - 1 = 1$.

Ahhoz, hogy egy $F(x)$ polinomról a Sturm-féle tétel alapján kiderítsük, van-e gyöke az u és v számok közötti intervallumon, elég a hányadossorozat ismerete, hiszen a

$$Q_1(u), \dots, Q_k(u), \quad Q_1(v), \dots, Q_k(v)$$

számokból az előjelsorozat már gyorsan kiszámolható.

Láttunk tehát olyan esetre példát, amikor az euklidészi algoritmus által előállított egész $(Q_1(x), \dots, Q_k(x), F_k(x))$ hányadosvektorra szükség volt.

Mennyi számolást igényel az euklidészi algoritmus? A tárgyalás további részében a polinomokra szorítkozunk. Az egész számok esetében minden hasonlóan megy, de valamivel több diszkusszióra van szükség.

Ha $F_0(x)$ foka kisebb, mint $F_1(x)$ foka, akkor nyilván $F_2(x) = F_0(x)$, $Q_1(x) = 0$, és a további sorozatban már az osztó foka mindig kisebb, mint az osztandóé. Tegyük fel ezért a következőkben, hogy $F_0(x)$ foka, n , nem kisebb, mint $F_1(x)$ foka, m . Hány algebrai műveletbe kerül az euklidészi algoritmus? Legyen a $Q_1(x)$, $\dots$, $Q_k(x)$ polinomok fokszáma rendre n_1 , $\dots$, n_k . Ha nincs szerencsénk, akkor előfordulhat, hogy e számok mindegyike 1, $F_k(x)$ foka nulla, és így $k=n$. Ekkor az algoritmusban n darab polinomosztás van. A hagyományos módszerrel egy osztás cn^2 műveletbe, és így az egész eljárás cn^3 műveletbe került. Ha a gyors osztást használjuk, akkor ezt lecsoríthatjuk $cn^2 \log n$ -re. Ennél lényegesen kisebb műveletszámot csak akkor várhatunk, ha óvatosabban fogalmazzuk meg, mit akarunk eredményül? Ha eredményül az egész $F_2(x)$, $\dots$, $F_k(x)$ sorozatot akarjuk, akkor nem sokat javíthatunk: ennek „kinyomtatása” is közel n^2 műveletbe kerül. Más lesz a helyzet, ha megelégszünk a

$$(Q_1(x), \dots, Q_k(x), F_k(x))$$

hányadosvektorral, hiszen ezen polinomok fokainak összege csak n lesz.

A következőkben kiderül, hogy a

$$Q_1(x), \dots, Q_k(x)$$

hányadossorozat ismeretében az $F_2(x)$, $\dots$, $F_k(x)$ polinomok bármelyike kiszámítható $n \log^2 n$ művelettel, tehát nem sokkal többel, mint ahány operáció a kinyomtatásához szükséges. Ez abból fog következni, hogy bármelyik (7)-beli $M_1 \dots M_i$ szorzatmátrixot, és így F_{i+1} -et is könnyű kiszámolni, hiszen

$$\begin{bmatrix} F_i \\ F_{i+1} \end{bmatrix} = M_1 \dots M_i \cdot \begin{bmatrix} F_0 \\ F_1 \end{bmatrix}.$$

Elfogadhatjuk tehát, hogy a Q_i , $\dots$, Q_k hányadossorozat tartalmazza az euklidészi algoritmus által szolgáltatott összes fontos információt. Ez a sorozat már n -nél kevesebb számmal jellemezhető, elvben tehát elképzelhető, hogy kiszámításához cn művelet elegendő. Strassen következő tétele azonban alsó korlátot ad:

- **Tétel (Strassen):** A Q_1 , $\dots$, Q_k hányadossorozat kiszámításához legalább $cn \log n$ nemlineáris művelet kell.

Lehet-e tényleg cn^2 -ről $cn \log n$ -re javítani? A német Lehmer vette észre a 30-as években, hogy $Q_i(x)$ kiszámításához elég $F_0(x)$ és $F_1(x)$ „elejének”, vagyis a néhány legmagasabb fokú tagnak az ismerete.

3.11. feladat: Legyenek a $Q_1, \dots, Q_k$ hányadossorozat fokszámai $n_1, \dots, n_k$. Mutassuk meg, hogy $Q_1, \dots, Q_j$ csak az F_0 és F_1 polinom első k_j tagjától függ, ahol

$$k_j = n_1 + 2n_2 + \dots + 2n_j.$$

Ezt a megfigyelést kiaknázva adott *Schönhage* egy olyan eljárást, ami optimálisnak tekinthető. Legyen r egy természetes szám, és i az a legnagyobb szám, amellyel még a $Q_1, \dots, Q_k$ polinomok közül az első i darab $n_1, \dots, n_i$ fokszámaira teljesül az

$$n_1 + n_2 + \dots + n_i \leq r$$

összefüggés.

- **Tétel (Schönhage):** A $Q_1, \dots, Q_i$ hányadossorozat és a hozzá tartozó $M_1 \cdot M_2 \cdot \dots \cdot M_i$ szorzatmátrix együttesen kiszámolható $O(\log^2 r)$ művelettel. Ha csak a nemlineáris műveleteket számoljuk, $O(\log r)$ művelet elegendő.

3.2.4.

Mátrixinvertálás

Strassen gyors mátrixszorzási módszere felhasználható arra is, hogy n^3 helyett $O(n^{2.81})$ lépésben végezzük el a lineáris algebra egy sor mátrixoperációját: lineáris egyenletrendszerek megoldását, mátrixinvertálást, ortogonalizálást stb. Lássuk, hogyan gyorsul fel az invertálás a gyors szorzások segítségével. (Ebben a pontban az eddigieknél több lineáris algebrai ismeretet használunk fel. A szükséges alapismeretek megtalálhatók pl. *Rózsa Pálnak* a Bevezetésben idézett művében.)

Egy A mátrix transzponáltját A' -vel, az x és y vektorok skalárszorzatát (x, y) -nal jelöljük. Egy $n \times n$ -es szimmetrikus B mátrixot, mint tudjuk, pozitív definitnek nevezünk, ha minden, 0-tól különböző x vektorra $(x, Bx) > 0$ teljesül. Ha az A mátrix nem szinguláris, akkor $A'A$ pozitív definit, mert $(x, A'Ax) = (Ax, Ax) > 0$, ha $x \neq 0$.

Schönhage ötlete alapján a mátrixinvertálást visszavezetjük pozitív definit mátrix invertálására és szorzásokra. Jelölje $L(n)$ az $n \times n$ -es mátrixok invertálásához, $K(n)$ a szorzásokhoz, $L^+(n)$ pedig az $n \times n$ -es pozitív definit mátrixok invertálásához szükséges műveletek számát. A

$$B = A'A, \quad A^{-1} = B^{-1}A'$$

képlet alapján tetszőleges mátrix invertálása visszavezethető pozitív definit mátrix invertálására és két szorzásra:

$$L(n) \leq L^+(n) + 2K(n). \quad (10)$$

A továbbiakban tehát csak a szimmetrikus és pozitív definit **B** mátrix invertálásával kell foglalkoznunk. Legyen $n=2m$. **B** felírható

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_1 & \mathbf{B}_2 \\ \mathbf{B}_2' & \mathbf{B}_3 \end{bmatrix}$$

alakban, ahol $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{B}_3$ $m \times m$ -es mátrixok. Nyilván $\mathbf{B}_1$ és $\mathbf{B}_3$ is pozitív definit.

Ha a **B** mátrix pozitív definit és **T** egy nemszinguláris mátrix, akkor pozitív definit lesz a $\mathbf{T}'\mathbf{B}\mathbf{T}$ mátrix is, hiszen $\mathbf{x} \neq 0$ -ból következik, hogy

$$(\mathbf{x}, \mathbf{T}'\mathbf{B}\mathbf{T}\mathbf{x}) = (\mathbf{T}\mathbf{x}, \mathbf{B}(\mathbf{T}\mathbf{x})) > 0.$$

Legyen $\mathbf{U} = \mathbf{B}_1^{-1}\mathbf{B}_2$, és

$$\mathbf{T} = \begin{bmatrix} \mathbf{I} & -\mathbf{U} \\ \mathbf{0} & \mathbf{I} \end{bmatrix},$$

ahol **I** az $m \times m$ -es egységmátrix. Szorzással egyszerűen ellenőrizhető, hogy

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{I} & \mathbf{U} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \quad (11)$$

és a $\mathbf{C} = \mathbf{T}'\mathbf{B}\mathbf{T}$ mátrix a következő alakot ölti:

$$\mathbf{C} = \begin{bmatrix} \mathbf{B}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_3 \end{bmatrix}, \quad \text{ahol} \quad \mathbf{C}_3 = \mathbf{B}_3 - \mathbf{B}_2'\mathbf{U}.$$

Mivel a **C** mátrix szimmetrikus és pozitív definit, ezért a $\mathbf{C}_3$ mátrix is az. **C**-t már könnyű invertálni:

$$\mathbf{C}^{-1} = \begin{bmatrix} \mathbf{B}_1^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_3^{-1} \end{bmatrix}. \quad (12)$$

Végül $\mathbf{B}^{-1} = \mathbf{T}\mathbf{C}^{-1}\mathbf{T}'$ felbontása a (11) és (12) képletek szerint a következőképpen alakul:

$$\mathbf{B}^{-1} = \begin{bmatrix} \mathbf{B}_1^{-1} + \mathbf{V}\mathbf{U}' & -\mathbf{V} \\ -\mathbf{V}' & \mathbf{C}_3^{-1} \end{bmatrix}, \quad \text{ahol} \quad \mathbf{V} = \mathbf{U}\mathbf{C}_3^{-1}. \quad (13)$$

Hány művelet szükséges $\mathbf{B}^{-1}$ kiszámításához a (13) képlet alapján? A következő műveleteket kell végezni $m \times m$ -es mátrixokkal:

$\mathbf{B}_1^{-1}$	1 invertálás
$\mathbf{U} = \mathbf{B}_1^{-1}\mathbf{B}_2$	1 szorzás
$\mathbf{C}_3 = \mathbf{B}_3 - \mathbf{B}_2'\mathbf{U}$	1 szorzás és 1 összeadás

C_3^{-1}	1 invertálás
$V = UC_3^{-1}$	1 szorzás
$B_1^{-1} + VU'$	1 szorzás és 1 összeadás

összesen: 2 invertálás, 4 szorzás és 2 összeadás. Tehát

$$L^+(2m) \leq 2L^+(m) + 4K(m) + 2m^2.$$

Innen és (10)-ből

$$L(2m) \leq 2L(m) + 4K(m) + 2K(2m) + 2m^2.$$

Ebből az egyenlőtlenségből, valamint a mátrixszorzásra vonatkozó 3.1.2. pontbeli eredményből most már az ott alkalmazotthoz hasonló módszerrel levezethetjük *Strassen* tételét:

- **Tétel:** $L(n) \leq cn^{\log_2 7}$.

3.12. feladat: Bizonyítsuk be *Strassen* tételét.

3.3.

Polinomok kiértékelése

Gyakori, hogy ki kell számolni egy adott többváltozós polinom helyettesítési értékét egy adott helyen. Ezt nevezzük a polinom-kiértékelés feladatának. Előfordul, hogy nem egy, hanem több polinom helyettesítési értékére van szükségünk.

A polinomszorzás 3.1.2. pontban tárgyalt problémájánál találkoztunk az $(a_0, \dots, a_{n-1})$, $(b_0, \dots, b_{n-1})$ sorozatok $(c_0, \dots, c_{2n-2})$ konvolúciójának fogalmával. A $c_0, c_1, \dots$ együtthatók másodfokú polinomjai az $a_0, a_1, \dots, b_0, b_1, \dots$ számoknak:

$$c_i = C_i(a_0, \dots, a_{n-1}, b_0, \dots, b_{n-1}) = a_0 b_i + \dots + a_i b_0.$$

A polinomszorzás esetében a tulajdonképpeni feladat a

$$C_0(a_0, \dots, b_{n-1}), \dots, C_{2n-2}(a_0, \dots, b_{n-1})$$

másodfokú polinomok kiértékelése. A skalárszorzás esetében is egy másodfokú polinom, a mátrixszorzás esetében pedig néhány skalárszorzat kiértékelésével van dolgunk.

Az is gyakran fordul elő, hogy egy polinomot több, különböző helyen akarunk kiértékelni. A polinomszorzás feladatának megoldása során pl. hasznosnak bizonyult a $P(x)$, $Q(x)$ polinomokat a $\lambda_0, \lambda_1, \dots, \lambda_{2n-2}$ helyeken kiértékelni.

3.3.1.

Pan alsó becslései

Tekintsünk egy egyváltozós polinomot:

$$P(x) = a_0 + \dots + a_n x^n.$$

Hogyan lehet ezt a legkevesebb művelettel kiértékelni? Ha a műveleteket úgy végezzük el, ahogy ki vannak jelölve, azaz előbb kiszámoljuk x^2 , x^3 , ..., x^n -et, akkor ez $n-2$ összeadást és $2n-1$ szorzást követel. A szorzások számát felére csökkenthetjük egy egyszerű átalakítással:

$$P(x) = a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + x a_n) \dots)).$$

Ez a múlt század óta ismert *Horner-eljárás*, a szorzások száma itt n .

Talán nem túlságosan meglepő, hogy se kevesebb szorzás, se kevesebb összeadás nem elegendő, mint amennyi a Horner-eljárásban van. Most a szükséges szorzások számával foglalkozunk. Az általunk vizsgált algoritmusok szimbólumokból és számokból kifejezéseket építenek fel. (A szimbólumok helyére az algoritmus minden egyes alkalmazása során természetesen tetszőleges számokat lehet helyettesíteni.)

Konkrét számok közötti műveleteknek az eredménye is konkrét szám, amit nem kell az algoritmus minden egyes alkalmazása során újból meghatározni: tárolhatjuk a „programmal” együtt. Ha azonban az

$$a_0 + a_1 x + \dots + a_n x^n$$

polinom kiszámítására akarunk algoritmust találni, akkor, mivel általános algoritmust keresünk, az együttthatókat szimbólumoknak kell tekinteni. Hagyjuk el az érdektelen első tagot: feladatunk ekkor a

$$G(x_1, \dots, x_n, y) = x_1 y + \dots + x_n y^n$$

polinom kiszámítása.

Megjegyzés: Természetesen sok konkrét polinom van, amit gyorsan ki lehet értékelni. Pl. az

$$y^n + \binom{n}{n-1} y^{n-1} + \dots + n y + 1 = (1+y)^n$$

polinomhoz $c \log n$ művelet is elég. Az általános esetben ilyen átalakítási trükkök nem segítenek.

Lássunk egy másik példát arra, hogy milyen, látszólag jelentéktelen dolgokon múlhat egy polinom kiszámolásának egyszerűsége. Emlékezzünk algebrai tanulmányainkból az

$$\begin{vmatrix} x_{11} & \dots & x_{1n} \\ \vdots & & \vdots \\ x_{n1} & \dots & x_{nn} \end{vmatrix}$$

determináns definíciójára! A másod- és harmadrendű determinánst kiírjuk:

$$\begin{vmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{vmatrix} = x_{11}x_{22} - x_{12}x_{21} ,$$

$$\begin{vmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{vmatrix} = x_{11}x_{22}x_{33} + x_{12}x_{23}x_{31} + x_{13}x_{21}x_{32} - \\ - x_{11}x_{23}x_{32} - x_{13}x_{22}x_{31} - x_{12}x_{21}x_{33}$$

A determináns az $x_{11}, x_{12}, \dots, x_{n-1}, x_{nn}$ változók egy polinomja. Az $1, 2, \dots, n$ számok minden $i_1, i_2, \dots, i_n$ permutációjához tartozik egy

$$x_{1i_1} x_{2i_2} \dots x_{ni_n}$$

tag, amelynek előjelét az határozza meg, hány számpár van az $i_1, i_2, \dots, i_n$ permutációban fordított sorrendben. (Az előjel akkor és csak akkor pozitív, ha ez a szám, a permutációban előforduló „inverziók” száma, páros.)

A kombinatorikában és a fizikában időnként használnak egy polinomot, amely nagyon hasonló a determinánshoz: ugyanazok a tagok fordulnak elő benne, de csupa pozitív előjellel. A neve: permanens, jelölése:

$$\text{Per} \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & & \vdots \\ x_{n1} & \dots & x_{nn} \end{bmatrix} .$$

A permanens kiszámítására senki sem ismer jobb általános módszert, mint az egyes tagok külön kiszámítását és összeadását. Mivel a tagok száma $n!$, ez reménytelenül sok munkát követel, ha n nem nagyon kis szám.

Legújabbban Valiant megmutatta, hogy az univerzális kereső feladatok (1. 2. fejezet) visszavezethetők egy n^{konstans} méretű 0/1 mátrix permanensének a kiszámítására. Így polinomiális algoritmus találása a permanens kiszámítására nem is várható. A determináns ellenben, mint ismeretes, különféle átalakítási szabályok segítségével viszonylag könnyen (n^3 , sőt Strassen módszerével $n^{2,8}$ számú lépésben) kiszámolható.

Térjünk vissza az általános polinomkiszámítási algoritmusokra. Négy alapműveletet, az összeadást (+), kivonást (−), szorzást (·) és osztást (÷) engedünk meg. Ezekkel tetszőleges $F(x_1, \dots, x_n, y)$ racionális (azaz négy alapművelet segítségével felépített) kifejezés kiszámítható.

Az áttekinthetőség kedvéért csak az $x_1, \dots, x_n$ szimbólumokkal végzett nemlineáris műveletekre figyelünk. Nem számoljuk tehát

1. azokat a műveleteket, amelyeket csak y -okat tartalmazó kifejezésekkel végzünk,
2. az összeadásokat, kivonásokat, számmal (és nem változóval) való szorzást, osztást.

Egy ilyen algoritmus bemenő adatai lehetnek tetszőleges $G_i(y)$ kifejezések, eredménye az $F(x_1, \dots, x_n, y)$ kifejezés. Szemléltetésére bemutattunk egy, az

$$F(x_1, x_2, y) = \frac{\frac{1}{y} + x_1(y^4 - y + 5)}{\frac{1}{2}x_2 + y^2} + \frac{1}{x_1 - y + 3y^3}$$

kifejezés kiszámolására szolgáló algoritmust.

$$\textcircled{1} \quad z_1 \leftarrow x_1 * (y^4 - y + 5)$$

$$2. \quad z_2 \leftarrow \frac{1}{y} + z_1$$

$$3. \quad z_3 \leftarrow \frac{1}{2} * x_2$$

$$4. \quad z_4 \leftarrow z_3 + y^2$$

$$\textcircled{5} \quad z_5 \leftarrow \frac{z_2}{z_4}$$

$$6. \quad z_6 \leftarrow x_1 - (y - 3y^3)$$

$$\textcircled{7} \quad z_7 \leftarrow \frac{1}{z_6}$$

$$8. \quad z_8 \leftarrow z_5 + z_7$$

Mint látható, most $F(x_1, x_2, y)$ -t lényegében a képlet által előírt módon építettük fel, ez azonban nem lett volna kötelező. Fő, hogy a végeredmény ugyanaz legyen. Ebben az algoritmusban csak három darab nemlineáris művelet volt — azok, amelyeket bekarikáztunk — és z_8 volt a végeredmény.

Ha egy algoritmus kevés nemlineáris műveletet tartalmaz, akkor a végeredmény igen speciális alakra hozható.

Ha minden művelet lineáris, akkor a végeredmény lineáris $x_1, \dots, x_n$ -ben, azaz

$$a_1x_1 + \dots + a_nx_n + b(y)$$

alakú, ahol $a_1, \dots, a_n$ valós számok, $b(y)$ pedig egy, csak y -t tartalmazó kifejezés. Megmutatjuk, hogy a végeredmény mindig lineáris $x_1, \dots, x_n$ -ben, ha nem is az egész értelmezési tartományon, csak

annak bizonyos feltételekkel meghatározott részén. E feltételek példánkban következőképpen találhatók meg: Válasszunk x_1, x_2, y számára olyan értékeket, hogy minden előforduló osztásban az osztó nullától különböző legyen. Legyen például $\bar{x}_1=1, \bar{x}_2=0, \bar{y}=1$.

Haladjunk ezután végig az algoritmus műveletein. Az 1. jelű műveletet számolni kell, írjuk fel most az

$$y^4 - y + 5 = \bar{y}^4 - \bar{y} + 5,$$

tehát az

$$y^4 - y + 5 = 4 \quad (*)$$

egyenletet! Ennek az egyenletnek a figyelembevételével számoljunk tovább:

$$z_1 = 4x_1, \quad z_2 = 4x_1 + 1/y, \quad z_3 = 2x_1 + 1/2y, \quad z_4 = 2x_1 + 1/2y + y^2.$$

Így minden kifejezés továbbra is lineáris marad x_1, x_2 -ben, egészen addig, míg az 5. jelű, bekarikázott művelethez érünk. Írjuk most fel az 5.-ben fellépő osztás nevezőjére, z_4 -re a

$$z_4(x_1, x_2, y) = z_4(\bar{x}_1, \bar{x}_2, \bar{y}),$$

azaz az

$$\frac{1}{2}x_2 + y^2 = 1 \quad (**)$$

egyenletet. A további számolásban ezt is figyelembe vesszük, s így minden lineáris marad a 7. műveletig.

Itt végül felírjuk az

$$x_1 - y + 3y^3 = 2 \quad (***)$$

egyenletet. Ha a $(*)-(***)$ egyenletek teljesülnek, akkor:

$$F(x_1, x_2, y) = \frac{1}{y} + x_1 + 1/3.$$

Tudjuk, hogy a $(*)-(***)$ egyenletek megoldhatók, hiszen az $\bar{x}_1=1, \bar{x}_2=0, \bar{y}=1$ értékrendszer kielégíti őket.

Ezt a konstrukciót általánosítva, kimondhatjuk:

- **Tétel:** Ha egy az $F(x_1, \dots, x_n, y)$ kifejezés kiszámítására szolgáló algoritmus m darab nemlineáris műveletet tartalmaz, akkor

felírhatunk egy megoldható, m darab egyenletet tartalmazó egyenletrendszert:

$$\begin{aligned} c_{11}x_1 + \dots + c_{1n}x_n &= d_1(y) \\ \vdots & \\ c_{m1}x_1 + \dots + c_{mn}x_n &= d_m(y) \end{aligned} \quad (1)$$

és egy $a_1x_1 + \dots + a_nx_n + b(y)$ kifejezést. Azokra az $x_1, \dots, x_n, y$ értékrendszerekre, melyek kielégítik (1)-et, teljesül, hogy

$$F(x_1, \dots, x_n, y) = a_1x_1 + \dots + a_nx_n + b(y). \quad (2)$$

Alkalmazzuk most ezt az eredményt az $x_1y + \dots + x_ny^n$ polinomot kiszámoló algoritmusokra!

- **Tétel (Pan):** Ha egy algoritmus az $x_1y + \dots + x_ny^n$ polinomot adja végeredményül, akkor legalább n darab nemlineáris műveletet kell tartalmaznia.

A tétel igazolására tegyük fel, hogy a szóban forgó algoritmusok között van olyan, melyben a számolt műveletek m száma kisebb, mint n . Ez a feltételezés ellentmondásra vezet. Ekkor ugyanis az (1) egyenletrendszerhez tartozó

$$\begin{aligned} c_{11}x_1 + \dots + c_{1n}x_n &= 0 \\ \vdots & \\ c_{m1}x_1 + \dots + c_{mn}x_n &= 0 \end{aligned}$$

homogén lineáris egyenletrendszerben az egyenletek száma kisebb, mint az ismeretlenek száma. Az ilyen rendszernek, mint tudjuk, van egy olyan $u_1, \dots, u_n$ megoldása, melyben nem minden u_i nulla. Legyen ezenkívül az (1) egyenletrendszer egy megoldása $v_1(y), \dots, v_n(y)$. Ez y minden olyan értéke mellett megoldás lesz, amikor $v_1(y), \dots, v_n(y)$ -ban, $d_1(y), \dots, d_m(y)$ -ban és $b(y)$ -ban az esetleges nevezők nem tűnnek el. Nevezzük ezeket az értékeket y reguláris értékeinek. Mivel az inhomogén lineáris egyenletrendszer egy megoldásához a megfelelő homogén egyenletrendszer bármely megoldásának konstanszorosát hozzáadva újra megoldást kapunk, a

$$v_1(y) + u_1t, \dots, v_n(y) + u_nt$$

kifejezések az y minden reguláris értéke és a t minden értéke mellett eleget tesznek az (1) egyenletrendszernek. Ugyanezekre az értékekre fennáll tehát (2) megfelelője:

$$\begin{aligned} [v_1(y) + u_1t]y + \dots + [v_n(y) + u_nt]y^n &= \\ = a_1[v_1(y) + u_1t] + \dots + a_n[v_n(y) + u_nt] + b(y). \end{aligned}$$

Átalakítva:

$$v_1(y)y + \dots + v_n(y)y^n + t(u_1y + \dots + u_ny^n) = \\ = a_1v_1(y) + \dots + a_nv_n(y) + b(y) + t(u_1a_1 + \dots + u_na_n).$$

A fenti egyenlőség csak úgy állhat fenn minden t -re, ha t együtt-hatója a bal és jobb oldalon ugyanaz. Ez pedig, lévén, hogy $u_1, \dots, u_n$ között legalább egy nem 0 szám van, nem lehet y minden reguláris értékére igaz. Abból a feltételezésből tehát, hogy az $x_1y + \dots + x_ny^n$ polinomot kiszámító algoritmus n -nél kevesebb nemlineáris műveletet tartalmaz, valóban ellentmondásra jutottunk.

Használjuk fel a bemutatott módszert még egy hasonló állítás bizonyítására! Azt állítjuk, hogy

$$x_1^2 + \dots + x_n^2$$

nem számolható ki n -nél kevesebb szorzással vagy osztással. Aki esetleg azt hinné, hogy ez nyilvánvaló, gondolkozzon azon, hogy az $x_1^2 - x_2^2$ kifejezést miért lehet mégis egyetlen szorzással előállítani, az $x_1^2 - x_2^2 = (x_1 + x_2)(x_1 - x_2)$ azonosság alapján! Megint tegyük fel, hogy van olyan algoritmus, amely csak $n-1$ számolt műveletet tartalmaz és eredménye $x_1^2 + \dots + x_n^2$. Ekkor találhatók olyan $u_1, \dots, u_n, a_1, \dots, a_n, v_1, \dots, v_n, b$ számok, ahol $u_1, \dots, u_n$ nem mindegyike 0, és t minden értékére teljesül a

$$(v_1 + u_1t)^2 + \dots + (v_n + u_nt)^2 = a_1(v_1 + u_1t) + \dots + a_n(v_n + u_nt) + b$$

egyenlőség. Átalakítva:

$$t^2(u_1^2 + \dots + u_n^2) = tc_1 + c_2,$$

ahol c_1 és c_2 konstansok.

Ez az azonosság csak úgy állhat fenn minden t -re, ha $u_1^2 + \dots + u_n^2 = 0$. Mivel $u_1, \dots, u_n$ közül nem mindegyik 0, innen ellentmondásra jutunk.

3.13. feladat: Mutassuk meg, hogy komplex számok körében az $x_1^2 + \dots + x_n^2$ négyzetösszeg $\frac{n}{2}$ szorzással is kiszámítható. Miért mond akkor csődöt a fenti bizonyítás?

3.14. feladat: *Pan* módszerével lássuk be, hogy ha össze kell szoroznunk egy $k \times n$ -es mátrixot és egy n hosszúságú vektort:

$$\begin{bmatrix} f_1 \\ \vdots \\ f_k \end{bmatrix} = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \vdots & \vdots \\ x_{k1} & \dots & x_{kn} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix},$$

azaz képeznünk kell az

$$f_i = x_{i1}y_1 + \dots + x_{in}y_n \quad (i = 1, \dots, k)$$

összegeket, akkor ehhez legalább mn nemlineáris műveletet kell végrehajtánunk.

Tanulságos összevetni a 2. feladat eredményét a korábban közölt *Strassen*-féle mátrixszorzási módszerrel. Hiszen ha képezni kell az

$$\begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & & \vdots \\ x_{n1} & \dots & x_{nn} \end{bmatrix} \begin{bmatrix} y_{11} & \dots & y_{1n} \\ \vdots & & \vdots \\ y_{n1} & \dots & y_{nn} \end{bmatrix}$$

szorzatot, ez tulajdonképpen n darab mátrixvektor szorzást jelent: minden i -re ki kell számítani az

$$\begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & & \vdots \\ x_{n1} & \dots & x_{nn} \end{bmatrix} \begin{bmatrix} y_{1i} \\ \vdots \\ y_{ni} \end{bmatrix}$$

szorzatot. És noha minden ilyen szorzat kiszámítása külön-külön n^2 nemlineáris műveletbe kerülne, *Strassen* módszerével együtt n^3 helyett $cn^{2,81}$ művelettel is kiszámíthatók.

A *Pan*-féle alsó becslések értéke sajnos, sohasem haladhatja meg a változók számát. Ha tehát egy feladat olyan bonyolult, hogy megoldása a változók számánál szemmel láthatóan sokkal több műveletet igényel, akkor ahhoz, hogy erről a tényről matematikailag is meggyőződjünk, más módszereket kell keresni. *Strassen* eredményei, melyek az euklidészi algoritmus, a tabellázás és az interpoláció esetében $n \log n$ -es alsó becslést adnak, ahol n a változók száma (l. 3.2.3. és 3.3.3. pont), már olyan technikán alapulnak, amelynek ismertetésére itt nem vállalkozhatunk.

A helyzet kissé emlékeztet arra, amit a „Gráfok diagnosztikája” részben tapasztalhattunk. Ott is csak olyan alsó becsléseket tudtunk adni a lépések számára, mint az adatok száma. (Mivel azonban a kérdések száma több se lehetett, mint az adatoké, ott az alsó becslések mindig azt állították, hogy az adott esetben „mindent meg kell kérdezni”).

3.3.2. Prekondicionálás

A gyors szorzás és gyors osztás eljárásait nem régen ismerik, és „kézi” számolásban nem is érdemes ezeket használni. A sokjegyű számok gyors szorzása mégsem okoz problémát a 17. század eleje óta. Ennek igen egyszerű oka van. Azóta vannak logaritmus-tábláink! Ezeket természetesen — különösen eleinte, kézi erővel — nehéz volt kiszámolni, de hányszorosan megtérült azóta a befektetett munka! Minden b számhoz kiszámolták $\log b$ és e^b értékét, és ha ezután két tetszőleges számot, b -t és c -t, össze kellett szorozni,

e művelet helyett csak $\log b$ -t kellett összeadni $\log c$ -vel. Az ilyen előkészítő jellegű számolást *prekondicionálásnak* nevezik.

Miért jobb logaritmustáblát csinálni, mint mondjuk szorzástáblát? Tegyük föl, hogy az 1 és 10 közötti számokat akarjuk 4 tizedesjegy pontossággal szorozni. Logaritmustáblában kb. $9 \cdot 10^4$ szám logaritmusát kell leírni, ami megvalósítható. Szorzástáblában $9 \cdot 10^4 \cdot 9 \cdot 10^4 \approx 8 \cdot 10^9$ szorzatot kellene tárolni, ami lehetetlen.

Először megmutatjuk annak előnyét, ha a skalárszorzást prekondicionálással végezzük.

Pan módszerével beláttuk, hogy az $\mathbf{x} = (x_1, \dots, x_n)$ és $\mathbf{y} = (y_1, \dots, y_n)$ vektorok

$$(\mathbf{x}, \mathbf{y}) = x_1 y_1 + \dots + x_n y_n$$

skalárszorzatának kiszámítása legalább n darab nemlineáris műveletet kíván. Itt is elképzelhető, azonban, hogy azok a műveletek, amelyeket egy-egy vektoron belül végzünk, kisebb súllyal esnek latba, és ezért első közelítésben nem kell őket számolni. Márpedig ekkor a skalárszorzáshoz már $n/2$ darab nemlineáris művelet is elég, amint ezt a következő átalakítás mutatja

$$x_1 y_1 + x_2 y_2 = (x_1 + y_2)(y_1 + x_2) - x_1 x_2 - y_1 y_2.$$

A jobb oldal második és harmadik tagjának kiszámítása prekondicionálásnak számít: az olyan szorzások száma, amelyekben a két tényező *különböző* vektorokból származik, csak feleannyi, ha a jobb oldali és nem a bal oldali elrendezést használjuk. Ennek az észrevételnek akkor lehet jelentősége, ha például sok különböző vektort kell tárolni, és ezek közül gyakran kell kettőt összeszorozni. Ilyenkor esetleg érdemes minden egyes $\mathbf{x}$ vektorhoz előre kiszámítani és vele együtt tárolni az

$$s(\mathbf{x}) = x_1 x_2 + x_3 x_4 + \dots$$

összeget is, mert akkor az $\mathbf{x}^{(i)}$ és $\mathbf{x}^{(j)}$ vektorok skalárszorzata az

$$(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (x_1^{(i)} + x_2^{(j)})(x_1^{(j)} + x_2^{(i)}) + (x_3^{(i)} + x_4^{(j)})(x_3^{(j)} + x_4^{(i)}) + \dots - s(\mathbf{x}^{(i)}) - s(\mathbf{x}^{(j)})$$

képlet alapján $n/2$ nemlineáris művelettel előállítható. Ennél többet a skalárszorzás esetén prekondicionálással sem lehet nyerni. $n/2$ nemlineáris műveletre mindenképpen szükség van. Ezt a szovjet *Belaga* bizonyította be. A prekondicionálás egy másik alkalmazási területe a polinomkiértékelés. Tegyük fel, hogy egy adott

$$P(x) = a_0 + a_1 x + \dots + a_n x^n$$

polinomot gyakran akarunk kiértékelni különböző x helyeken. A későbbi számolások meggyorsítása érdekében esetleg hajlandók

vagyunk egyszer nagyobb áldozatot hozni: bizonyos dolgokat előre kiszámolni. Ehhez természetesen csak az együttthatók $a_0, \dots, a_n$ értékeit használhatjuk fel. Most az érdekel minket, hogy mennyit lehet az előkészítéssel, a polinom prekondicionálásával nyerni!

Feladatunkat szabatosan ismét olyan algoritmusok segítségével fogalmazhatjuk meg, amilyenek a 3.2.1. pontban szerepeltek. Bemenő adatoknak $a_0, \dots, a_n, x$ és tetszőleges számok vehetők. A négy alpműveletet lehet használni. Csak az olyan műveleteket számoljuk azonban, amelyekben x -et tartalmazó kifejezés nem-lineáris módon szerepel. Más szóval, csak a következő fajta utasításokat számoljuk:

$$1. \quad z_i \leftarrow z_j * z_k,$$

ahol z_j és z_k is függ x -től.

$$2. \quad z_i \leftarrow \frac{z_j}{z_k}, \text{ ahol } z_k \text{ függ } x\text{-től.}$$

Megmutatjuk, hogy van olyan algoritmus $P(x)$ kiértékelésére, amely így csak $n/2$ nemlineáris műveletet tartalmaz. Alkalmas prekondicionálással tehát a szorzások fele megtakarítható.

Indulásként próbáljuk a skalárszorzásnál észrevett átalakítási lehetőséget a polinomszorzásra is kamatoztatni: A

$$Q(x) = b_0 + b_1x + b_2x^2 + \dots + b_{m-1}x^{m-1}$$

polinomot felfoghatjuk mint két vektor,

$$(b_0, b_1, \dots, b_{m-1}) \quad \text{és} \quad (1, x, x^2, \dots, x^{m-1})$$

skalárszorzatát. Foglaljuk össze most is páronként a tagokat, de nem sorban, hanem az elsőt az utolsóval, másodikat az utolsó előttivel stb. (tegyük fel, hogy m páros):

$$\begin{aligned} Q(x) &= (b_0 + b_{m-1}x^{m-1}) + (b_1x + b_{m-2}x^{m-2}) + (b_2x^2 + b_{m-3}x^{m-3}) + \dots = \\ &= \left[(b_0 + x^{m-1})(1 + b_{m-1}) + (b_1 + x^{m-2})(x + b_{m-2}) + \dots - \frac{m}{2} x^{m-1} \right] - \\ &\quad - (b_0b_{m-1} + b_1b_{m-2} + \dots + b_{\frac{m}{2}-1}b_{\frac{m}{2}}). \end{aligned}$$

Mit nyertünk ezzel az átalakítással? Számoljuk össze a szorzásokat!

1. $x^2, x^3, \dots, x^{m-1}$ előállításához $m-2$ darab szorzás,
2. $b_0b_{m-1} + \dots + b_{\frac{m}{2}-1}b_{\frac{m}{2}}$ előállításához $\frac{m}{2}$ darab szorzás,

3. a szögletes zárójelben levő tag kiszámításához $\frac{m}{2}$ darab szorzás szükséges.

A 2. alatti műveleteket nem kell számolni: ezek előre elvégezhetők. Az 1. alatti műveleteket számolni kell, de van egy kellemes tulajdonságuk, a $b_0, b_1, \dots, b_n$ számok nem fordulnak elő bennük. Ezért, ha több polinomot kellene ugyanezen az x helyen kiszámolni, e műveletek jelentősége csökkenne. A számolt műveletek száma $m + \frac{m}{2}$. Ha k darab prekondicionált n -ed fokú polinomot kellene ugyanezen az x helyen kiértékelni, ez a szám $k\frac{m}{2} + m$ lenne.

Most e tulajdonságot felhasználjuk egy

$$P(x) = a_0 + a_1x + \dots + a_nx^{n-1}$$

$(n-1)$ -ed fokú polinom kiértékeléséhez, ahol $n \leq km$.

Osszuk a tagokat m -es csoportokba:

$$P(x) = Q_0(x) + Q_1(x)x^m + Q_2(x)x^{2m} + \dots + Q_{k-1}(x)x^{(k-1)m},$$

ahol

$$Q_0(x) = a_0 + \dots + a_{m-1}x^{m-1},$$

$$Q_1(x) = a_m + \dots + a_{2m-1}x^{m-1},$$

$$Q_{k-1}(x) = a_{(k-1)m} + \dots + a_{km-1}x^{m-1}.$$

Amint láttuk, $Q_0(x), \dots, Q_{k-1}(x)$ kiszámítása legfeljebb

$$\frac{km}{2} + m$$

számolt műveletbe került. Mivel x^m már megvan, $P(x)$ -et ezután a $Q_0(x), \dots, Q_{k-1}(x)$ együtthatókból további $k-1$ művelettel meghatározhatjuk. Az egész számolás így legfeljebb

$$\frac{km}{2} + m + k$$

műveletet követel, ahol k és m csak a $km \geq n$ feltételnek vannak alávetve. Ha pl.

$$k = m = \lceil \sqrt{n} \rceil,$$

akkor a műveletszám $n/2 + 2\lceil \sqrt{n} \rceil$ -nel becsülhető.

Van tehát egy algoritmusunk, amely a $P(x)$ polinom értékét — prekondicionálással — $n/2 + c\sqrt{n}$ számolt művelettel állítja elő.

A polinom prekondicionálása abban áll, hogy előre ki kell számolni és vele együtt tárolni a k darab

$$\begin{aligned} s_0 &= a_0 a_{m-1} + a_1 a_{m-2} + \dots \\ s_1 &= a_m a_{2m-1} + a_{m+1} a_{2m-2} + \dots \\ &\vdots \\ s_{k-1} &= a_{(k-1)m} a_{km-1} + a_{(k-1)m+1} a_{km-2} + \dots \end{aligned}$$

összeget.

Ismeretes olyan (valamivel több algebrai fogalmat felhasználó) algoritmus is, amely a prekondicionálás után már csak

$$\left\lceil \frac{n}{2} \right\rceil + 2$$

műveletet igényel. Azt is bebizonyították (*Belaga*), hogy $n/2$ -nél kevesebb művelet nem elegendő.

Végül megjegyezzük, hogy a számítógépek elemi függvényeket (pl. $\sin x$ -et) szolgáltató szubrutinjai rendszerint egy közelítő polinomot számolnak. Ezt azonban nem mindig könnyű felismerni, mert a program a polinomot már prekondicionált formában tartalmazza. A nagyobb pontosság is követelheti, hogy a polinomot ne eredeti formájában írjuk fel.

3.3.3.

Tabellázás és fordítottja, az interpoláció

Előfordulhat, hogy a $P(x) = a_0 + a_1 x + \dots + a_n x^n$ polinomnak sok különböző helyen felvett értékére van szükség: szeretnénk elkészíteni a

$$P(u_1), \dots, P(u_k)$$

táblázatot, ahol $u_1, \dots, u_k$ adott számok. A feladatot meg lehet oldani úgy, hogy $P(u_1), \dots, P(u_k)$ értékét *Horner* módszerével külön-külön kiszámoljuk. Ez kn darab nemlineáris műveletet kíván. Az előző pontban tárgyalt prekondicionálással ezt kb. $kn/2$ -re lehet leszorítani.

Felmerül a kérdés, nem lehetne-e nyerni azon, ha az egész táblázatot egyszerre számítjuk ki? Hátha bizonyos eredmények egyszerre több u_i esetén is hasznosak? Most megmutatjuk, hogy ha $k \geq n$, akkor a $P(u_1), \dots, P(u_k)$ táblázatot már

$$k \log n$$

nemlineáris művelettel és (ha mindent számolunk)

$$k \log^2 n$$

művelettel előállíthatjuk. Az egyszerűség kedvéért a $k=n$ esetre szorítkozunk.

- **Tétel:** Az n -ed fokú $P(x)$ polinom n helyen való együttes kiértékeléséhez $cn \log n$ nemlineáris, és összesen $cn \log^2 n$ művelet elegendő.

Ez az eljárás nem javítható lényegesen, mert Strassen bebizonyította a következő tételt.

- **Tétel:** A

$$P(u_1), \dots, P(u_n)$$

értékek kiszámítása legalább $n \log(n-1)$ nemlineáris műveletet kíván.

Hasonlítsuk össze az első Tételt a gyors Fourier-transzformálás algoritmusával! Ott $P(\varepsilon), P(\varepsilon^2), \dots, P(\varepsilon^n)$ kiszámításához $cn \log n$ művelet kellett. Most tetszőleges $u_1, \dots, u_n$ számokra is kiszámoljuk $P(x)$ értékét, $cn \log^2 n$ lépésben. Ezt azonban, mint látni fogjuk, éppen a gyors Fourier-transzformációnak (ill. a segítségével gyorsan végrehajtható szorzásnak) köszönhetjük.

Rátérünk a $P(u_1), \dots, P(u_n)$ -et kiszámító algoritmusra. Ezekkel az értékekkel egyidőben az

$$I(x, u_1, \dots, u_n) = (x - u_1)(x - u_2) \dots (x - u_n)$$

polinom együtthatóit is kiszámítjuk. Az $I(x, u_1, \dots, u_n)$ polinom n -ed fokú, benne x^n együtthatója 1, és gyökei az $u_1, \dots, u_n$ számok. (Ezek a tulajdonságok egyértelműen meg is határozzák a polinomot.) A szokásos alakban felírva:

$$I(x, u_1, \dots, u_n) = x^n - \sigma_1 x^{n-1} + \sigma_2 x^{n-2} + \dots,$$

ahol

$$\sigma_1 = u_1 + u_n,$$

$$\sigma_2 = u_1 u_2 + u_1 u_3 + \dots + u_{n-1} u_n,$$

$$\vdots$$

$$\sigma_n = u_1 u_2 \dots u_n$$

az $u_1, \dots, u_n$ számok ún. elemi szimmetrikus polinomjai. Eljárásunk rekurzív. Tegyük fel, hogy már tudjuk, mit kell tenni, ha $P(x)$ foka kisebb, mint az m szám. Most definiáljuk tovább az el-

járást olyan $P(x)$ polinomokra, amelyeknek foka kisebb vagy egyenlő, mint $2m$. A $P(x)$ -et esetleg nulla együtthatójú tagokkal kiegészítve, feltehetjük, hogy $P(x)$ foka $n=2m$, azaz

$$P(x) = a_0 + a_1x + \dots + a_{2m}x^{2m}.$$

Ki akarjuk számítani $P(x)$ értékét az

$$u_1, \dots, u_{2m}$$

helyeken. Osszuk ezeket a helyeket két részre: $u_1, \dots, u_m$ és $u_{m+1}, \dots, u_{2m}$ csoportokra. Képezzük az

$$I(x, u_1, \dots, u_m) \quad \text{és} \quad I(x, u_{m+1}, \dots, u_{2m})$$

polinomokat (az eljárás m -re már ismert!). Osszuk el a $P(x)$ polinomot az $I(x, u_1, \dots, u_m)$ polinommal:

$$P(x) = I(x, u_1, \dots, u_m)Q_0(x) + R_0(x),$$

és az $I(x, u_{m+1}, \dots, u_{2m})$ polinommal:

$$P(x) = I(x, u_{m+1}, \dots, u_{2m})Q_1(x) + R_1(x),$$

ahol az $R_0(x)$, $R_1(x)$ maradékpolinomok foka kisebb, mint m . Mivel $I(x, u_1, \dots, u_m)$ értéke nulla az $u_1, \dots, u_m$ helyeken, ezért

$$P(u_1) = R_0(u_1), \dots, P(u_m) = R_0(u_m).$$

Hasonlóan,

$$P(u_{m+1}) = R_1(u_{m+1}), \dots, P(u_{2m}) = R_1(u_{2m}).$$

Ilyen módon megtehetjük, hogy ahelyett, hogy a $2m$ -ed fokú $P(x)$ polinomot értékelnénk ki az $u_1, \dots, u_{2m}$ helyeken, az m -nél kisebb fokú $R_0(x)$, $R_1(x)$ polinomokat értékeljük ki az $u_1, \dots, u_m$, ill. az $u_{m+1}, \dots, u_{2m}$ helyeken. Az eredmény ugyanaz. Az m -nél kisebb fokú polinomokra viszont az eljárás már definiálva van.

Hátravan még $I(x, u_1, \dots, u_{2m})$ kiszámítása. Ez egyszerű, mert

$$I(x, u_1, \dots, u_{2m}) = I(x, u_1, \dots, u_m)I(x, u_{m+1}, \dots, u_{2m}).$$

Becsüljük meg a felhasznált műveletszámot! Jelöljük $M(n)$ -nel két, legfeljebb n -ed fokú polinom összeszorzásához, $D(n)$ -nel pedig az osztáshoz szükséges műveletek számát, $T(n)$ -nel a $P(u_1), \dots, P(u_n)$ és $I(x, u_1, \dots, u_n)$ kiszámításához szükséges műveletek együttes számát. A fenti eljárás alapján

$$T(2m) \leq 2T(m) + 2D(2m) + M(m). \quad (3)$$

Tudjuk, 3.1. és 3.2. szakaszból, hogy

$$M_{\text{nemlin}}(n) \leq c_1 n, \quad D_{\text{nemlin}}(n) \leq c_2 n,$$

$$M_{\text{össz}}(n) \leq c_3 n \log n, \quad D_{\text{össz}}(n) \leq c_4 n \log n$$

(c_1, c_2, c_3, c_4 állandók), ezért (3)-ból a szükséges nemlineáris, ill. az összes szükséges műveletek számára a következő felső becslést kapjuk:

$$T_{\text{nemlin}}(n) \leq \frac{c_1 + 4c_2}{2} n \log n,$$

$$T_{\text{össz}}(n) \leq \frac{c_3 + 4c_4}{2} n \log^2 n.$$

Az algoritmus tehát valóban igen gyorsan dolgozik.

A tabellázás műveletének fordítottja az interpoláció: adottak az $u_0, \dots, u_n, v_0, \dots, v_n$ számok, ($u_0, \dots, u_n$ mind különböző), határozzuk meg azt az n -ed fokú $P(x)$ polinomot, amely eleget tesz a

$$P(u_0) = v_0, \dots, P(u_n) = v_n$$

feltételeknek. (Más szóval: illesszünk egy n -ed fokú görbét az

$$(u_0, v_0), \dots, (u_n, v_n)$$

koordinátapárokra). Ezzel a feladattal már találkoztunk a polinomszorzásról szóló fejezetben. Láttuk, hogy az

$$a_0 + a_1 u_0 + \dots + a_n u_0^n = v_0$$

$$\vdots$$

$$a_0 + a_1 u_n + \dots + a_n u_n^n = v_n$$

lineáris egyenletrendszerből a $P(x)$ polinom ismeretlen $a_0, \dots, a_n$ együtthatói valóban meghatározhatók, mégpedig olyan műveletekkel is, amelyek lineárisak $v_0, \dots, v_n$ -ben. Most azonban $u_0, \dots, u_n$ -et is változónak kell tekinteni, ezért a felmerülő műveletek általában nem lesznek lineárisak. Ennek az interpolációs feladatnak speciális esete nemcsak a Fourier-transzformáció inverze, hanem lényegében az $I(x, u_1, \dots, u_n)$ polinom kiszámításának feladata is. Hiszen a szóban forgó polinomot a

$$P(u_1) = 0, \dots, P(u_n) = 0$$

feltételek határozzák meg. Strassen következő tétele alsó becslést ad a szükséges nemlineáris műveletek számára.

- **Tétel:** $I(x, u_1, \dots, u_n)$ kiszámításához legalább $n \log n$ nemlineáris műveletre van szükség.

Megmutatjuk, hogy ennyi elég is, nemcsak $I(x, u_1, \dots, u_n)$ kiszámításához, de az általános interpolációs feladat megoldásához is. Célunk tehát olyan algoritmus megadása, amely az $u_0, \dots, u_n, v_0, \dots, v_n$ értékek alapján kiszámítja a $P(x)$ polinom $a_0, \dots, a_n$ együtthatóit. Bebizonyítjuk a következő tételt:

- **Tétel:** Az interpolációs feladat $cn \log n$ nemlineáris művelettel, és (ha mindent számolunk) $cn \log^2 n$ művelettel megoldható.

Legyen

$$D(x) = I(x; u_1, \dots, u_n),$$

$$D_i(x) = (x - u_1) \dots (x - u_{i-1})(x - u_{i+1}) \dots (x - u_n) = \frac{D(x)}{x - u_i};$$

és legyen

$$d_i = D_i(u_i) = (u_i - u_1) \dots (u_i - u_{i-1})(u_i - u_{i+1}) \dots (u_i - u_n).$$

A $P(x)$ polinomot Lagrange képlete adja meg:

$$(*) \quad P(x) = \frac{v_1}{d_1} D_1(x) + \dots + \frac{v_n}{d_n} D_n(x).$$

Valóban, ha x helyébe u_i -t helyettesítünk, akkor ebben az összegben az i -edik kivételével minden tag 0, az i -edik tag pedig

$$\frac{v_i}{d_i} D_i(u_i) = v_i.$$

Feladatunk tehát a $(*)$ polinom együtthatóinak gyors kiszámítása.

1. Első lépésként kiszámoljuk az $I(x; u_1, \dots, u_n) = D(x)$ polinomot, $n \log n$ lépésben, az előző pontban ismerttetett módon. Részeredményként megkapjuk az $I(x; u_1, \dots, u_{\frac{n}{2}}), I(x; u_{\frac{n}{2}+1}, \dots, u_n)$

polinomokat is, valamint az összes többi, ezekből hasonló felzéssel kapható polinomokat. (Feltehetjük, hogy $n = 2^k$. Ha n nem ilyen alakú, az utána következő első ilyen alakú számmal dolgozunk.)

2. Második lépésként kiszámoljuk a $d_1, \dots, d_n$ számokat. Ehhez a következő észrevételt használjuk: A Taylor-formula szerint

$$D(x) = D(u_i) + D'(u_i)(x - u_i) + F_i(x)(x - u_i)^2,$$

ahol $F_i(x)$ egy polinom. Mivel $D(u_i) = 0$, innen

$$D_i(x) = \frac{D(x)}{x - u_i} = D'(u_i) + (x - u_i)F_i(x),$$

$$d_i = D_i(u_i) = D'(u_i).$$

Tehát d_i nem más, mint a $D'(x)$ polinom u_i helyen felvett értéke. A $D'(x)$ polinom az előző pont eljárásával kiértékelhető az $u_1, \dots, u_n$ helyeken $n \log n$ lépésben, ezért a $d_1, \dots, d_n$ számokat is $n \log n$ lépésben megkaptuk.

3. Hátravan még a $w_i = v_i/d_i$ számok ismeretében a (*) képletbeli $P(x)$ kiszámítása. Feledkezzünk meg most $w_1, \dots, w_n$ eredeti jelentéséről, tekintsük e számokat is a feladat bemenő adatainak: tetszőleges $u_1, \dots, u_n, w_1, \dots, w_n$ számokra kerestetik a (*) képlet által megadott $P(x; u_1, \dots, u_n; w_1, \dots, w_n)$ polinom. Legyen $n = 2m$. Osszuk két részre az (*) képletbeli összeadandókat, és emeljünk ki az első és a második részből is annyi tényezőt, amennyit csak tudunk! Az alábbi összefüggést kapjuk:

$$P(x; u_1, \dots, u_{2m}; w_1, \dots, w_{2m}) =$$

$$(**) = I(x; u_{m+1}, \dots, u_{2m})P(x; u_1, \dots, u_m; w_1, \dots, w_m) +$$

$$+ I(x; u_1, \dots, u_m)P(x; u_{m+1}, \dots, u_{2m}; w_{m+1}, \dots, w_{2m}).$$

Az $I(x; u_1, \dots, u_m)$, $I(x; u_{m+1}, \dots, u_{2m})$ polinomokat már az 1. lépésben kiszámoltuk. Ez a képlet tehát a $2m$ -ed fokú $P(x)$ polinom kiszámítását visszavezeti két, hasonló típusú, m -ed fokú polinom kiszámítására és két polinomszorzásra. Ha $K(n)$ a $P(x; u_1, \dots, u_n; w_1, \dots, w_n)$ kiszámításához — az 1. lépés után — szükséges műveletszám, akkor (**) szerint

$$K(2m) \leq 2K(m) + 2M(m).$$

Innen újra azt kapjuk, hogy a nemlineáris műveletek száma $c \cdot n \log n$, az összes $c \cdot n \log^2 n$.

Összegezve: Láttuk, hogy a tabellázás és az interpoláció feladatainak speciális esete az oda-, ill. visszafele történő *Fourier*-transzformáció. Ez a feladat egy ügyes fogás segítségével $cn \log n$ művelettel elvégezhető. A *Fourier*-transzformáció lehetővé teszi a gyors polinomszorzást. A gyors szorzás lehetővé teszi a gyors osztást. A gyors szorzás és osztás segítségével viszont az általános interpolációs és tabellázási feladat is $cn \log^2 n$ lépésben megoldható.

Meglepő, hogy ezek az eredmények mind az utóbbi 10 évben születtek, hiszen a bennük szereplő fogalmak 150 évvel ezelőtt ugyanúgy ismertek voltak, mint ma. Természetesen a fejezetünk elején hangoztatott megjegyzés itt is érvényes: lehet, hogy a műveletek számának aszimptotikájában mutatott fölényük ellenére is, a bemutatott eljárások gyakorlati alkalmazása még hosszú ideig várat magára.

3.4.

Párhuzamos számolás

Eddig az algebrai és kombinatorikai algoritmusok nehézségét csak a szükséges műveletek számával mértük, ezt igyekeztünk minimalizálni. Sok esetben annak van nagyobb jelentősége, hogy az egész számolás gyorsan véget érjen, „kerül amibe kerül”. Ennek érdekében hajlandók vagyunk több számítógépet beállítani, vagy akár az adott feladat végrehajtására új számítógépet építeni, amelybe minden művelet „be van drótozva”, és amely minden pillanatban annyi műveletet végez, amennyi a számolás adott szakaszában egymástól függetlenül csak lehetséges.

Valószínűleg nem minden algoritmust lehet úgy átalakítani, hogy több számítógéppel gyorsabban lehessen kiszámítani, mint eggyel (az algoritmust „párhuzamosítani”). Ehhez rendszerint a feladat alapos átgondolására, eredeti ötletekre van szükség. Az is érdekes kérdés, hogy adott feladat megoldó algoritmusát mennyire lehet és mennyire érdemes párhuzamosítani? Az e könyvben közölt algoritmusok nagy részét maximálisan lehet párhuzamosítani. A gyors polinomszorzás algoritmus $\log n$, a gyors polinomosztás $\log^2 n$ szakaszra bontható, és az egyes szakaszokon belül a műveleteket párhuzamosan lehet végezni, kb. n műveletet egy szakaszban.

Ugyanakkor az euklidészi algoritmust (amely ugyancsak $n \log^2 n$ műveletet igényel), aligha lehet az általános esetben n -nél kevesebb szakaszban elvégezni: itt a számolás tehát nem gyors, viszont az idő nagy hányadában csak kevés aktív kapacitást köt le.

A 2. fejezet ún. nehéz feladatai párhuzamosan működő számítógépekkel rövid idő alatt megoldhatók lennének. Ezek a feladatok ui. próbálgatással megoldhatók, és az egyes esetek végigpróbálása egymástól függetlenül történhet. Ezért, ha elég (exponenciálisan) sok gépet tudunk párhuzamosan működtetni, akkor megtehetjük, hogy egy gráf összes színezését egyidejűleg vizsgáljuk.

Ebben a részben megmutatjuk, hogy az *algebrai formulák kiszámítása* szükség esetén *maximálisan párhuzamosítható*.

A számolásnak egy részét akkor kell mindenképpen előbb elvégezni mint a többi, ha ez a rész sok olyan eredményt ad, amit a későbbiekben is fel kell használni. Ha *minden részeredményre csak egyszer van szükség*, akkor a számolásról egyszerű képet alkotunk. Szorítkozzunk a négy alpműveletre (összeadás, kivonás, szorzás és osztás). A hatványozást többszörös szorzásnak tekintjük.

A felírás alapján az

$$\frac{x+y}{3-yx} - \frac{1}{\frac{y(x+y)}{3-yx} + 2} \quad (1)$$

képlet kiszámításához a következő műveletsorra van szükség:

$$\begin{array}{ll} z_1 \leftarrow x + y & z_7 \leftarrow y * x \\ z_2 \leftarrow y * x & z_8 \leftarrow 3 - z_7 \\ z_3 \leftarrow 3 - z_2 & z_9 \leftarrow \frac{z_6}{z_8} \\ z_4 \leftarrow \frac{z_1}{z_3} & z_{10} \leftarrow z_9 + 2 \\ z_5 \leftarrow x + y & z_{11} \leftarrow 1/z_{10} \\ z_6 \leftarrow y * z_5 & z_{12} \leftarrow z_4 - z_{11} \end{array}$$

Itt a $z_1, \dots, z_{11}$ részeredmények mindegyikét csak egyszer használjuk fel. Vegyük észre, hogy a részeredmények részformulákként lokalizálhatók. Ha egy részeredményt többször is felhasználhatunk, akkor példánkban sok egyszerűsítés kínálkozik.

3.15. feladat: Lássuk be, hogy a következő műveletsor ugyanazt a végeredményt adja, mint (2):

$$\begin{array}{ll} z_1 \leftarrow x + y & z_5 \leftarrow y * z_4 \\ z_2 \leftarrow y * x & z_6 \leftarrow z_5 + 2 \\ z_3 \leftarrow 3 - z_2 & z_7 \leftarrow \frac{1}{z_6} \\ z_4 \leftarrow \frac{z_1}{z_3} & z_8 \leftarrow z_4 - z_7. \end{array} \quad (3)$$

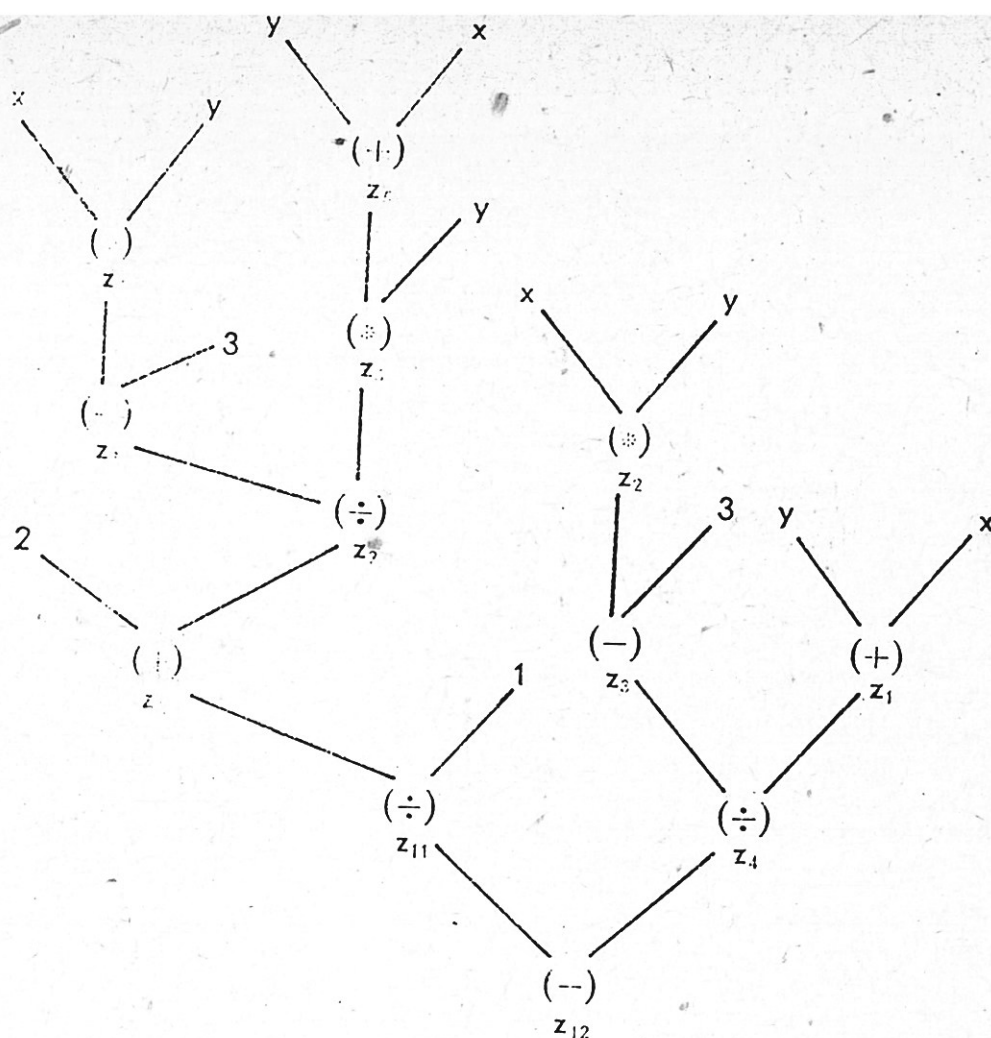
A legtöbb képletben azonban nem számíthatunk sok, a fentihez hasonló egyszerűsítésre, ezért az előbbi primitívebb eljárás sem nevezhető mesterkéltnek.

A példa alapján általánosabban is kimondhatjuk: egy számolás, amelyben minden részeredményt csak egyszer használunk fel, egyértelműen megfelel annak a képletnek, amelyet kiértékel.

A (2) számolás alapján a 17. ábrán látható, fa alakú diagramot rajzolhatnánk fel.

A diagramból az (1) formula egyértelműen rekonstruálható. A diagram elágazási pontjai, azaz a $z_1, \dots, z_{12}$ változók megfelelnek az (1) formula részformuláinak. A számolásban végrehajtandó műveletek száma megegyezik a formulában előforduló műveleti jelek számával.

Mit értünk azon, hogy egy formula kiszámolása párhuzamosítható? A párhuzamos számolást a következő formában lehet elképzelni: a t -edik szakaszban a $z_1^{(t)}, z_2^{(t)}, \dots$ értékeket egyszerre



17. ábra

számoljuk ki a korábbi szakaszokban kapott értékekből és a bemenő adatokból. Egy művelet tehát

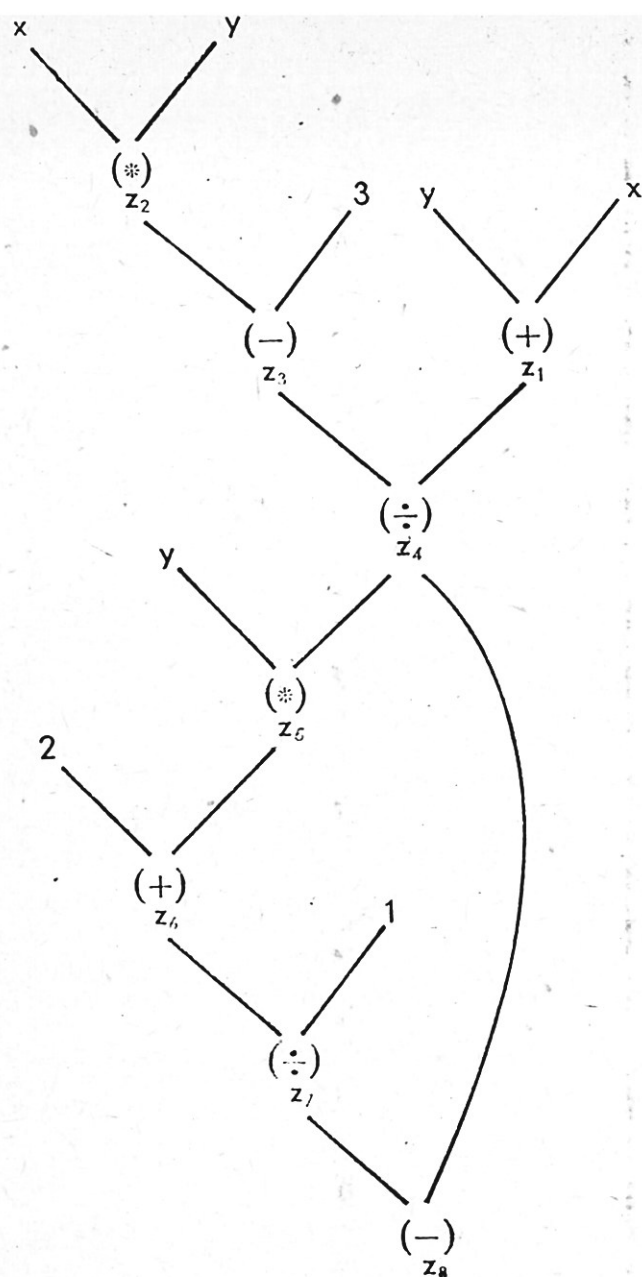
$$z_i^{(t)} \leftarrow \varphi(u, v)$$

alakú lesz, ahol φ az alapműveletek valamelyike, u és v pedig vagy a bemenő adatok valamelyike, vagy $z_j^{(s)}$, ahol s kisebb, mint t . A szemléletesség kedvéért az azonos t -hez tartozó képleteket egy sorba írhatjuk.

Próbáljunk a (2) számolásban szakaszokat bevezetni!

- | | | |
|-------------|----------------------|-------------|
| 1. szakasz: | z_1, z_2, z_5, z_7 | kiszámítása |
| 2. szakasz: | z_3, z_6, z_8 | kiszámítása |
| 3. szakasz: | z_4, z_9 | kiszámítása |
| 4. szakasz: | z_{10} | kiszámítása |
| 5. szakasz: | z_{11} | kiszámítása |
| 6. szakasz: | z_{12} | kiszámítása |

Mivel az egy szakaszhoz tartozó műveletek egyszerre végezhetők el, az egész számolás 6 lépésben véget érhet. Amíg a 17. ábra elrende-



18. ábra

zéséhez ragaszkodunk, addig többet nem is érhetünk el. A szakaszok minimális száma ugyanis nyilván egyenlő annak a leghosszabb útnak a hosszával, amely a hálózat egy legfelső elemétől a legalsó eleméig vezet.

A 18. ábrán láthatjuk a (3) számoláshoz tartozó diagramot.

A lehető legrövidebb idő, amely alatt ez a számolás elvégezhető, nyilván újra egyenlő lesz a felülről lefelé vezető leghosszabb út hosszával. Érdekes, hogy ez a szám itt 7, eggyel több, mint az előbbi, nyilvánvalóan gazdaságtalanabb számolásban. Ha tehát elég apparátus áll rendelkezésünkre, a 17. ábrabeli elrendezés gyorsabb számolást tesz lehetővé, mint a 18. ábrabeli. (Mondanunk sem kell, hogy (2) számolásban a 17. ábra leghosszabb útjának növelése nélkül is hajtható végre egyszerűsítés: pl. z_1 helyett z_5 és z_3 helyett z_8 használatával.)

Ha adott az elvégzendő műveletek L száma, akkor a szakaszok számát úgy igyekezzük csökkenteni, hogy egy-egy szakaszban minél több műveletet helyezünk el. Az egy szakaszban elhelyezhető műveletek számának azonban felső határa van. Tegyük fel, hogy az eljárás T darab szakaszból áll. Akkor az utolsó szakaszban csak egy művelet szerepel: a végeredmény, $z_1^{(T)}$ kiszámítása:

$$z_1^{(T)} = \varphi^{(T)}(u, v).$$

Az utolsó előtti szakaszban legfeljebb 2 művelet lehet, hiszen $z_1^{(T)}$ kiszámítása legfeljebb két adatot igényel:

$$z_1^{(T-1)} = \varphi_1^{(T-1)}(u_1, v_1)$$

$$z_2^{(T-1)} = \varphi_2^{(T-1)}(u_2, v_2)$$

és így tovább: a $(T-k)$ -edik szakaszban legfeljebb 2^k művelet van. Ezért a szakaszok T száma és a műveletek L száma között a következő egyenlőtlenség áll fenn:

$$1 + 2 + \dots + 2^{T-1} = 2^T - 1 \geq L,$$

azaz

$$T \geq \log_2 (L + 1).$$

Most már megfogalmazhatjuk, mit értsünk azon, hogy egy adott, (1) alakú A számolás maximálisan párhuzamosítható. Ezen azt értjük, hogy ha a számolás L műveletet tartalmaz, található egy olyan B számolás, amely ugyanazt az eredményt adja, mint A , de benne a szakaszok száma nem sokkal több, mint $\log L$. A négy alapműveletet tartalmazó formulákkal valóban így áll a helyzet.

- **Tétel (Brent):** Legyen $F(x_1, \dots, x_k)$ egy formula, amelyben a műveleti jelek száma L . Ekkor $F(x_1, \dots, x_k)$ T szakaszban kiszámolható, ahol

$$T \leq 1 + c \log L.$$

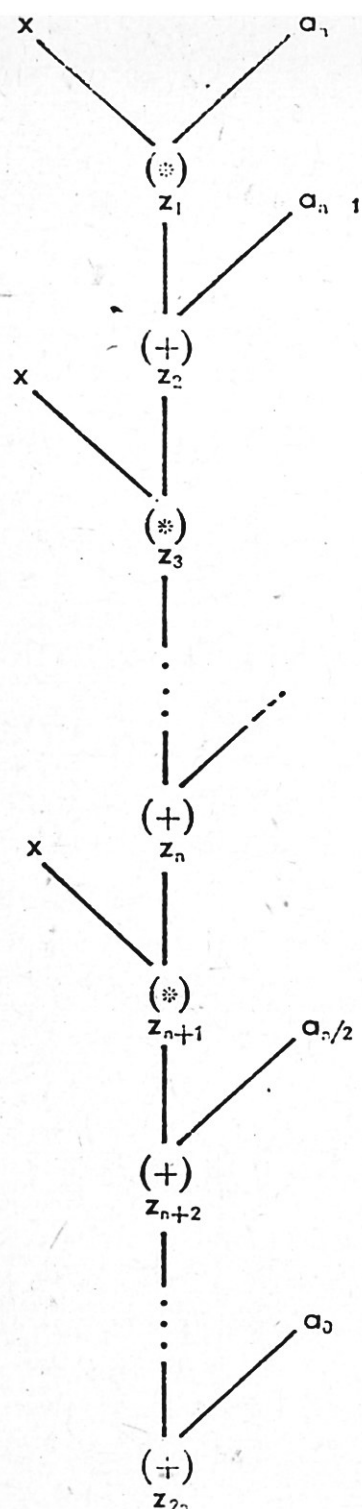
Szemléltetésül alkalmazzuk a tételt az

$$F(x, a_0, \dots, a_n) = a_0 + a_1x + \dots + a_nx^n$$

n -ed fokú polinomra. Ez persze még nem olyan formula, amelyre tételünk vonatkozik, mert hatványozás is van benne. Írjuk át ezért Horner módszere szerint (e módszerről úgyis tudjuk, hogy nagyon gazdaságos):

$$F(x, a_0, \dots, a_n) = a_0 + x\{a_1 + x[a_2 + \dots x(a_{n-1} + xa_n) \dots]\}.$$

Ebben a formulában $2n$ művelet van: n szorzás és n összeadás. A számolás diagramját a 19. ábra mutatja. A szakaszok száma



19. ábra

a lehető legnagyobb, $2n$. A tétel azt állítja, hogy olyan módszert is találhatunk $F(x, a_0, \dots, a_n)$ kiszámítására, amelyben a szakaszok száma csak $c \log n$.

Hogyan lehetne ezt a hosszú műveletsort, ahol mindegyik művelet az előző eredményére vár, kisebb és egymástól független részekre tördelni? Szeretnénk például z_n kiszámolását a maradék műveletektől függetleníteni. Ez valóban megoldható, mert

$F(x, a_0, \dots, a_n)$ egyszerű módon függ z_n -től (tegyük fel, hogy n páros):

$$F(x, a_0, \dots, a_n) = a_0 + x\{a_1 + x[\dots(a_{n/2} + xz_n)\dots]\} = \\ = a_0 + x\{a_1 + x[\dots(a_{n/2-1} + xa_{n/2})\dots]\} + x^{n/2}z_n.$$

Ha tehát z_n kiszámításával párhuzamosan számoljuk a

$$G(x, a_0, \dots, a_{n/2}) = a_0 + x[a_1 + \dots(a_{n/2-1} + xa_{n/2})\dots]$$

és

$$H(x) = \underbrace{x \cdot x \cdot \dots \cdot x}_{n/2\text{-szer}}$$

kifejezéseket, akkor utána már az

$$F(x, a_0, \dots, a_n) = G(x, a_0, \dots, a_{n/2}) + H(x)z_n \quad (4)$$

képlet alapján F -et további 2 művelettel megkaphatjuk. Nyilvánvaló, hogy a számolás ilyen „félbetörését” a G és H kifejezések, valamint z_n előállítása esetében is végrehajthatjuk. A félbetörés addig folytatható, amíg a formulák hossza egynél nagyobb. Minden félbetörés után a (4)-beli két művelet kell az „összeragasztáshoz”. Mivel a félbetörések száma kb. $\log n$, így a végül kapott eljárásban a szakaszok száma kb. $2 \log n$ lesz.

A számolás párhuzamosítása (és ezzel *Brent* tételének bizonyítása) hasonlóan megy az általános esetben is.

3.16. feladat: Mutassuk meg, hogy ha $F(x_1, \dots, x_n)$ L darab műveletet tartalmaz, és x_1 az F -ben csak egy helyen szerepel, akkor F az

$$F(x_1, \dots, x_n) = \frac{x_1 G_1(x_2, \dots, x_n) + G_2(x_2, \dots, x_n)}{x_1 G_3(x_2, \dots, x_n) + G_4(x_2, \dots, x_n)}$$

alakba írható, ahol $G_1, \dots, G_4$ kiszámítása is legfeljebb $2L$ műveletet kíván.

3.17. feladat: Mutassuk meg, hogy az F formula $z_1, \dots, z_L$ diagramjában található olyan z_1 pont, amelyre

(a) elhagyva a z_1 feletti pontokat, a diagram megmaradó pontjainak száma nem több, mint $L/3$.

(b) ha közvetlenül z_1 felett pl. a z_j és z_k pontok állnak: $z_1 = \varphi(z_j, z_k)$, akkor se a z_j , se a z_k feletti pontok száma nem haladja meg $2L/3$ -at.

3.18. feladat: Az 1. és 2. feladat eredményeinek felhasználásával bizonyítsuk be *Brent* tételét.

3.19. feladat: Mutassuk meg, hogy a $\neg, \wedge, \vee$ logikai jelekből felépített kifejezések számolása is maximálisan párhuzamosítható.

3.5. Moduláris algoritmusok

Az igen nagy számokkal végzendő elemi műveleteket nemcsak elméleti érdeklődésből vizsgáljuk. Igaz, hogy nincs olyan gyakorlati feladat, amelyben 40 értékes jegy pontosságú adatokra lenne szükség. Annál több viszont az olyan eljárás, amelyben az egyes lépések során végrehajtott *kerekítések* szerencsétlen esetben az eredményt teljesen használhatatlanná teszik: előfordulhat, hogy csak a részeredmények rendkívül pontos kiszámolása segít.

A sokjegyű számokkal végzett műveleteket jelentősen fel lehet gyorsítani a 3.1. és 3.2. szakaszok algoritmusaival. A számítógépek egész jelenlegi szervezettsége kizárja azonban annak lehetőségét, hogy pl. 500 jegyű számokkal sok műveletet végezzünk. Ebben a fejezetben a probléma egy másfajta megközelítéséről, a moduláris, algoritmusokról lesz szó. Ezek olyan algoritmusok, amelyek néhány egyszerű számelméleti tény felhasználásával egyrészt megkerülik a részeredmények korlátlan növekedésének problémáját, másrészt sok esetben lehetővé teszik a *párhuzamos számolást* (lásd a 3.4. szakaszt).

A moduláris algoritmusokat máris sikerrel alkalmazzák, és — ellentétben sok egyéb, a könyvben ismertetett újdonsággal — ez esetben bátoríthatunk mindenkit, hogy olyan feladatoknál, ahol hasznosnak tűnnek, kísérletezzon velük.

3.5.1. A kínai maradéktétel

Az ebben a pontban közölt számelméleti ismereteket részletesebb formában közlik a tankönyvek, valamint sorozatunk egy következő, a számelmületről szóló kötete.

„Két páros szám összege páros. Páratlan számok összege páros. Páros és páratlan szám összege páratlan. Ha két szám közül az egyik páros, akkor a szorzatuk páros; páratlan számok szorzata páratlan.”

Látjuk, hogy b és c számok összegének párosságát meghatározza az összeadandók párossága. Azt a tényt, hogy b és c párossága egyforma, így jelöljük:

$$b \equiv c \pmod{2}.$$

Kimondva: b kongruens c -vel modulo 2.

Az idézőjelek közé tett szöveget lefordíthatjuk a kongruenciák nyelvére: ha

$$b_0 \equiv c_0 \quad \text{és} \quad b_1 \equiv c_1 \pmod{2},$$

akkor

$$b_0 + b_1 \equiv c_0 + c_1 \quad \text{és} \quad b_0 b_1 \equiv c_0 c_1 \pmod{2}.$$

Más szóval: a mod 2 kongruenciákat egymással összeadhatjuk vagy összesorozhatjuk, és így újra kongruenciákat kapunk. Ugyanez igaz a kivonásra is. *Kongruenciákkal tehát bizonyos határok között úgy lehet számolni, mint az egyenlőségekkel.*

Minden szám kongruens 0-val vagy 1-gyel modulo 2. Ha páros, akkor 0-val, ha páratlan, akkor 1-gyel kongruens.

Kongruenciák nemcsak a 2 „modulusra” definiálhatók. Ha m egész szám, akkor a

$$b \equiv c \pmod{m}$$

reláció azt jelenti, hogy a b és c számok m -mel osztva ugyanazt a maradékot adják, azaz, a különbségük osztható m -mel.

$$1526 \equiv 16 \pmod{10},$$

mert 10-zel osztva mindkét szám 6-ot ad maradékul. Más szóval a $b \equiv c \pmod{m}$ reláció azt jelenti, hogy van egy olyan k egész szám, mellyel

$$b - c = km.$$

Nyilván minden b szám kongruens a $0, 1, \dots, m-1$ számok valamelyikével modulo m . Ez utóbbi számot a b szám *legkisebb pozitív maradékának* nevezzük. Sokszor a legkisebb pozitív maradék helyett a *legkisebb abszolút értékű maradékot* részesítjük előnyben. Ha az m szám páros, akkor a legkisebb abszolút értékű maradék a

$$-\frac{m}{2} + 1, -\frac{m}{2} + 2, \dots, \frac{m}{2} - 1, \frac{m}{2}$$

számok, ha m páratlan, akkor a

$$-\frac{m-1}{2}, -\frac{m-1}{2} + 1, \dots, \frac{m-1}{2}$$

számok közül kerül ki. Az egymással kongruens számok ilyen módon m darab osztályba, az m szerinti maradékosztályokba sorolhatók.

Tetszőleges m szerinti kongruenciákra is könnyű az összeadási és szorzási tulajdonságokat ellenőrizni:

ha $b_0 \equiv c_0$ és $b_1 \equiv c_1 \pmod{m}$, akkor

$$b_0 + b_1 \equiv c_0 + c_1, \quad b_0 b_1 \equiv c_0 c_1 \pmod{m},$$

speciálisan: egy kongruencia mindkét oldalához hozzáadható, ill. belőle levonható ugyanaz a szám, és mindkét oldal beszorozható ugyanazzal a számmal. E tulajdonságok segítségével már elég szabadon tudunk kongruenciákat „átrendezni”. Pl. könnyen megválaszolhatjuk azt a kérdést, hogy milyen x számokra teljesül az

$$5x + 3 \equiv 6x - 8 \pmod{11}$$

kongruencia. Rendezzünk át:

$$(6 - 5)x \equiv 3 + 8, \quad \text{azaz } x \equiv 11 \equiv 0 \pmod{11}.$$

A megoldások tehát éppen a 11-gyel osztható számok. Látjuk, hogy az ismeretlent tartalmazó kongruenciákat sokszor az egyenletekhez hasonlóan lehet „megoldani”. Egyelőre komoly akadályt jelent azonban, hogy nem tudunk „osztani”.

Ha a

$$3x \equiv 5 \pmod{11} \tag{1}$$

„egyenletet” is meg tudnánk oldani, akkor a megoldást joggal nevezhetnénk az $5/3$ törtszám megfelelőjének, a 11-re vonatkozó maradékok között. Hogy mikor lehet az (1) típusú egyenleteket megoldani és hogyan, arról szól a következő számelméleti tétel.

- **Tétel:** *Ha a és b és m számok relatív prímek, akkor a $0, 1, \dots, m-1$ maradékok között van pontosan egy olyan c , melyre*

$$bc \equiv 1 \pmod{m}.$$

A c számot b reciprokanak (inverzének, az $1/b$ megfelelőjének) tekinthetjük, modulo m .

Ezt a számot könnyen meg is lehet határozni. Az euklidészi algoritmusról szóló fejezetben láttuk, hogy ez az eljárás mellékeredményként olyan u és v számokat is szolgáltat, amelyekkel

$$(b, m) = ub + vm.$$

(Ez az ottani (8) egyenlőség megfelelője.)

Ha b és m relatív prímek, akkor innen

$$1 \equiv ub \pmod{m},$$

azaz az u szám lesz a b reciproka modulo m .

Lássuk a tétel alkalmazását az (1) kongruenciára! Némi keresgélés után rátalálhatunk a 4 számra, ami a 3 inverze:

$$3 \cdot 4 \equiv 12 \equiv 1 \pmod{11},$$

és így (1) mindkét oldalát 4-gyel beszorozva azt kapjuk, hogy

$$x \equiv 4 \cdot 3x \equiv 4 \cdot 5 \equiv 9 \pmod{11},$$

azaz (1) megoldása 9, ez az $5/3$ megfelelője modulo 11.

Ha m prímszám, akkor minden b számhoz relatív prím, amely nem osztható m -mel. Ekkor tehát minden olyan b számnak, amely nem kongruens 0-val modulo m , van reciproka modulo m .

$$\text{Pl. } 2 \cdot 6 \equiv 1, \quad 3 \cdot 4 \equiv 1, \quad 5 \cdot 9 \equiv 1, \quad 7 \cdot 8 \equiv 1 \quad 10 \cdot 10 \equiv 1 \pmod{11}.$$

Egy m prímszám szerinti maradékok között tehát van összeadás, kivonás, szorzás és nem 0 maradékkal való osztás.

A fenti tulajdonságok hasznossága különösen akkor szembe-tűnő, ha egy hosszú műveletsor végeredményéről szeretnénk tudni, melyik maradékosztályba tartozik m szerint.

Például: $88 \cdot 76 - 65 \cdot 92$ maradékát mod 3 kiszámolhatjuk a nagy számokon végzett műveletek megkerülésével:

$$88 \equiv 1, \quad 76 \equiv 1, \quad 65 \equiv 2, \quad 92 \equiv 2 \pmod{3}, \text{ ezért}$$

$$88 \cdot 76 - 65 \cdot 92 \equiv 1 \cdot 1 - 2 \cdot 2 \equiv -3 \equiv 0 \pmod{3}.$$

Mivel ilyen módon könnyen juthatunk a számolás végeredményére vonatkozó információhoz, esetleg hajlandók vagyunk a végeredmény maradékát néhány különböző modulusra nézve is kiszámolni:

$$88 \cdot 76 - 65 \cdot 92 \equiv 8 \cdot 6 - 5 \cdot 2 \equiv 8 \pmod{10}.$$

Arra a kérdésre, hogy milyen módon hasznosíthatjuk az így szerzett információkat, a következő híres tétel válaszol.

- **Kínai maradéktétel:** *Tegyük fel, hogy az $m_1, m_2, \dots, m_k$ számok páronként relatív prímek, azaz semelyik kettőnek sincs közös osztója. Ekkor*

a) *Az $m_1, \dots, m_k$ szerinti maradékok meghatározzák az*

$$M = m_1 m_2 \dots m_k$$

szorzat szerinti maradékokat is: azaz, ha a

$$b \equiv c \pmod{m_1}$$

$$b \equiv c \pmod{m_2}$$

$$\vdots$$

$$b \equiv c \pmod{m_k}$$

kongruenciák mind teljesülnek, akkor fennáll a

$$b \equiv c \pmod{M}$$

kongruencia is.

b) Minden $b_1, b_2, \dots, b_k$ számsorozathoz van pontosan egy olyan M -nél kisebb szám, melyre

$$b \equiv b_1 \pmod{m_1}$$

$$b \equiv b_2 \pmod{m_2}$$

$\vdots$

$$b \equiv b_k \pmod{m_k}$$

Ezt a b számot ki is lehet számítani (l. alább).

A tétel lehetővé teszi, hogy az M -nél kisebb (vagy $M/2$ -nél kisebb abszolút értékű) számokat az $m_1, \dots, m_k$ szerinti maradékaikkal egyértelműen reprezentáljuk. A reprezentáció nem sokkal gazdaságosabb, mint a tízes, vagy kettes számrendszerbeli. Ha az m_1 -nél kisebb számokat kettes számrendszerben írjuk fel, akkor b -nek m_1 szerinti legkisebb pozitív maradéka egy legfeljebb $\log_2 m_1$ hosszúságú kettes számrendszerbeli szám, b_1 . Hasonlóképpen $b_2, \dots, b_k$ egy-egy $\log_2 m_2, \dots, \log_2 m_k$ hosszúságú bináris számmal írható le. A tétel b) része szerint viszont a $b_1, \dots, b_k$ maradéksorozat, már meghatározza a b számot. Ennek a sorozatnak a leírásához csak

$$\log_2 m_1 + \dots + \log_2 m_k = \log_2 (m_1 \dots m_k) = \log_2 M$$

darab bináris számjegy kell, nem több, mint amennyire amúgy is szükség van az M -nél kisebb b szám felírásához. Ha pl. $m_1, \dots, m_{100}$ száz darab prímszám 10^6 és 10^7 között, akkor a száz darab $b_1, \dots, b_{100}$ maradékkal (mindegyik legfeljebb 7 jegyű tízes számrendszerbeli szám, tehát a számítógép egy rekeszében elfér) együttesen már 600 jegyű számokat lehet egyértelműen reprezentálni.

Ennek a reprezentációnak az az előnye, hogy a maradékokon az egyszerű algebrai műveleteket külön-külön lehet végezni. Ha a b számot a $b_1, \dots, b_{100}$ sorozat, a c számot a $c_1, \dots, c_{100}$ sorozat reprezentálja, akkor a bc számot reprezentálja pl. a

$$b_1 c_1, \dots, b_{100} c_{100}$$

sorozat. (Hogy a reprezentáló számok újra a legkisebb maradékok legyenek, a $b_1 c_1$ számot el lehet osztani m_1 -gyel, és a maradékot írni a helyére. Hasonlóan járunk el a $b_2 c_2, \dots, b_{100} c_{100}$ számokkal.)

Legyünk azonban óvatosak! A bc szorzatot csak addig határozzák meg $m_1, \dots, m_k$ szerinti maradékai, amíg maga is kisebb, mint M .

Fontos kérdés, hogyan lehet a számok közönséges, tízes számrendszerbeli reprezentációjáról a fenti, maradékokkal való reprezentációjára való áttérést és az onnan való visszatérést gyakorlatilag megvalósítani? „Odafelé” az út egyszerű: ahhoz, hogy a b szám m_1 szerinti legkisebb maradékát megtudjuk, el kell osztani b -t m_1 -gyel. Nehezebb feladat a b számot a $b_1, \dots, b_k$ maradékokból visszakapni.

Keressük b -t a következő formában:

$$b \equiv c_1 + c_2 m_2 + c_3 m_1 m_2 + \dots + c_k m_1 m_2 \dots m_{k-1} \pmod{M}, \quad (2)$$

ahol $0 \leq c_1 < m_1, \dots, 0 \leq c_k < m_k$.

Ekkor $c_1, \dots, c_k$ -ra a következő egyenleteknek kell teljesülni:

$$\begin{aligned} b_1 &\equiv c_1 \pmod{m_1} \\ b_2 &\equiv c_1 + c_2 m_1 \pmod{m_2} \\ &\vdots \\ b_k &\equiv c_1 c_2 m_1 + \dots + c_k m_1 \dots m_{k-1} \pmod{m_k}. \end{aligned} \quad (3)$$

Ezek az egyenletek könnyen megoldhatók. Tegyük fel, hogy $1 \leq i < j \leq k$. Mivel az m_i és m_j számok relatív prímek, az euklidészi algoritmussal megtalálható az az u_{ij} szám, amely az m_i reciproka mod m_j :

$$0 < u_{ij} < m_j, \quad u_{ij} m_i \equiv 1 \pmod{m_j}.$$

Ezeket az u_{ij} számokat érdemes (ha van hely) egyszer és mindenkorra előre kiszámolni, amikor az $m_1, \dots, m_k$ modulusok mellett döntünk és elraktározni. Most már meg tudjuk oldani a (3) kongruenciákat:

$$\begin{aligned} c_1 &\equiv b_1 \pmod{m_1} \\ c_2 &\equiv u_{12}(b_2 - c_1) \pmod{m_2} \\ c_3 &\equiv u_{13}u_{23}(b_3 - c_1 - c_2 m_1) \pmod{m_3} \\ &\text{stb.} \end{aligned}$$

Az eddigi összes számolást még a viszonylag kicsi $m_1, \dots, m_k$ szerinti maradékokkal végezhetjük. Csak az utolsó lépésben, a

$$\begin{aligned} b &= c_1 + c_2 m_1 + \dots + c_k m_1 \dots m_k = \\ &= c_1 + m_1 \{ c_2 + m_2 [c_3 + \dots (c_{k-1} + m_{k-1} c_k) \dots] \} \end{aligned}$$

kiszámolásakor kell nagy számokkal dolgozni.

Ezt az utolsó lépést azonban sok esetben felesleges elvégezni. A $c_k, \dots, c_1$ sorozatnak ugyanis már megvan a tízes számrendszerbeli reprezentáció sok hasznos tulajdonsága.

Hasonlítsuk csak össze a két felírást:

$$b = d_s \cdot 10^s + d_{s-1} \cdot 10^{s-1} + \dots + d_1 \cdot 10 + d_0, \quad 0 \leq d_i < 10,$$

$$b = c_k m_{k-1} \dots m_1 + c_{k-1} m_{k-2} \dots m_1 + \dots + c_2 m_1 + c_1 \quad 0 \leq c_i < m_i.$$

Ha a b és b' számokat (2) alakban írjuk fel, akkor nagyságuk könnyen összehasonlítható, közelségük ellenőrizhető, a b szám kerekíthető stb.

Összefoglalás:

Tegyük fel, hogy egész számokon el kell végezni egy számítást, amelyben csak összeadás, kivonás és szorzás szerepel. Ennek egy lehetséges végrehajtási módja a következő:

1. Megbecsüljük, mekkorára nőhet fel az eredmény, majd választunk olyan $m_1, \dots, m_k$ számokat, melyek

- a) relatív prímek,
- b) szorzatuk, $M = m_1 \dots m_k$ legalább kétszer akkora, mint az eredmény nagysága,
- c) egyenként kb. egy rekeszt foglalnak el, így az elemi műveleteket gyorsan lehet velük végezni.

2. Minden egyes b bemenő adathoz kiszámoljuk a $b_1, \dots, b_k$ számokat, amelyek rendre a b legkisebb maradékai az $m_1, \dots, m_k$ modulusokra nézve.

3. Ugyanazokat a műveleteket, melyeket az eredeti számolás követel, most modulo m_1 hajtjuk végre. Ez azt jelenti, hogy minden művelet végrehajtása után külön kiszámítjuk az eredmény legkisebb maradékát modulo m_1 és azzal számolunk tovább. Így végül megkapjuk a számolás végeredményének maradékát mod m_1 . Hasonlóképpen járunk el a mod $m_2, \dots, \text{mod } m_k$ maradékokkal.

4. Ismerjük tehát a b' eredmény $b'_1, \dots, b'_k$ maradékait. Kiszámoljuk a b' -höz tartozó $c'_1, \dots, c'_k$ számokat. Végül, ha szükséges, előállítjuk a b' számot. Ha nem törekszünk abszolút pontosságra, akkor a $c'_1, \dots, c'_k$ számok közül néhány „kisebb helyiértékű” el is hagyhatunk. (Legnagyobb helyiértékű a c'_k .)

A moduláris algoritmusok legnagyobb hátránya az, hogy egészen $c'_1, \dots, c'_k$ kiszámításáig semmiféle információnk nincs a részeredmények nagyságáról, az approximációk mértékéről stb. Nem lehet észrevenni a túlsordulást, ezért előre biztosnak kell lenni abban, hogy nem is léphet fel.

Ha az elvégzendő számolásban osztás is van, de a végeredmény egész szám, akkor célszerű az $m_1, \dots, m_k$ számokat prímeknek választani. A következő pontban példát láthatunk ilyen eljárásra.

3.5.2.

Lineáris egyenletrendszer pontos megoldása

A moduláris algoritmusok legfontosabb alkalmazási területének az olyan algebrai feladatok tűnnek, ahol (az ismert megoldási eljárásban) a részeredmények jegyeinek száma nagyra nőhet, kerekíteni pedig nem szabad.

Annak, hogy kerekíteni nem szabad, kétféle oka lehet. Egyik, hogy a végeredmény egész szám, és a számunkra érdekes információt nem is feltétlenül a nagysága hordozza (hanem pl. az, hogy páros-e). Ez leginkább algebrai vagy számelméleti feladatoknál fordul elő. Ha például az euklidészi algoritmust közvetlenül akarjuk többváltozós polinomokra alkalmazni, akkor ilyen jellegű nehézségekkel találkozunk.

Vannak olyan elméleti fizikai feladatok is, ahol egy lineáris egyenletrendszer megoldásai egész számok, és teljes pontossággal van rájuk szükség.

A másik lehetséges ok az, hogy az adott eljárás mellett kis kerekítés is nagy torzításhoz vezethet az eredményben. Nem kell feltétlenül erőltetett példákra gondolni. Tekintsünk egy

$$Ax = b,$$

kiírva

$$\begin{aligned} a_{11}x_1 + \dots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{n1}x_1 + \dots + a_{nn}x_n &= b_n \end{aligned} \tag{4}$$

lineáris egyenletrendszert! Az a feladat, hogy „oldjuk meg (4)-et”, sokszor még nem nevezhető jól megfogalmazottnak. Hiszen előfordulhat, hogy a megoldás nem egyértelmű, és az is, hogy nincs megoldás. Helyesebb talán úgy megfogalmazni, hogy meg kell állapítani, megoldható-e (4), és ha igen, fel kell írni maximális számú független megoldást? Sajnos, még ezt a kérdést sem nevezhetjük értelmesnek. A (4) rendszer bemenő adatai, az

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}$$

mátrix és a

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

vektor rendszerint csak közelítően ismertek. Márpedig, ha sok megoldás van, akkor a $\mathbf{b}$ vektor akármilyen kis változtatása már olyan egyenlethez vezet, amelynek nincs megoldása. Ez a két eset tehát a közelítő adatok alapján nem választható szét egyértelműen. Ha viszont az $\mathbf{A}$ mátrixot változtatjuk meg kissé, akkor az új egyenletnek majdnem biztos, hogy pontosan egy megoldása lesz!

Még azokban az esetekben is, amikor a megoldás egyértelmű, előfordulhat, hogy a bemenő adatok jelentéktelen (a pontossági határokon bőven belül eső) változtatása a megoldás óriási arányú változásához vezet. Lássunk most olyan példát, amikor az ok nem vezethető vissza arra, hogy a „megoldhatatlanság — több megoldás” határon vagyunk. (A példa *Bahvalovtól* származik.) Mint tudjuk, ha az $\mathbf{A}$ mátrix determinánsa nem nulla, akkor a megoldás egyértelmű. Legyen most

$$\mathbf{A} = \begin{bmatrix} 1 & & & & & \\ 2 & 1 & & & & \\ & 2 & 1 & & & \\ & & 2 & \ddots & & \\ & & & 2 & & \\ & & & & 1 & \\ & & & & 2 & 1 \end{bmatrix}$$

($\mathbf{A}$ be nem töltött helyeken nullák állnak.) Ekkor $\mathbf{A}$ determinánsának értéke 1. Próbáljuk megoldani az $\mathbf{Ax} = \mathbf{b}$ egyenletet! Kiírva:

$$\begin{aligned} x_1 &= b_1 \\ 2x_1 + x_2 &= b_1 \\ &\vdots \\ 2x_{n-1} + x_n &= b_n. \end{aligned}$$

A formális megoldás egyszerű:

$$\begin{aligned} x_1 &= b_1 \\ x_2 &= b_2 - 2b_1 \\ &\vdots \\ x_n &= b_n - 2b_{n+1} + 4b_{n-2} - \dots + (-2^{n-1})b_1. \end{aligned}$$

Másképp írva:

$$x_n = (-2^{n-2})(-2b_1 + b_2) + (-2^{n-4})(-2b_3 + b_4) + \dots$$

Tegyük fel, hogy

$$b_1 \approx 1/2, \quad b_2 \approx 1, \quad b_3 \approx 1/2, \quad b_4 \approx 1, \quad \dots$$

Ekkor tehát

$$x_n \approx \begin{cases} 0, & \text{ha } n \text{ páros} \\ 1/2, & \text{ha } n \text{ páratlan.} \end{cases}$$

x_n tehát egy viszonylag kis szám, és nem várnánk óriási ingadozásokat attól, hogy b_1 értékét nem tudjuk teljes pontossággal. Márpedig, ha b_1 értéke csak 10^{-7} pontossággal ismert, és $n=100$, akkor x_n hibája

$$2^{99} \cdot 10^{-7} > 10^{22}.$$

Még ha a bemenő adatok pontosak is, x_n kiszámolása közben olyan nagy számok léphetnek fel, amelyeket a számítógép rekeszeinek korlátozott nagysága miatt kénytelenek lennének kerekíteni. Az eredmény ugyanilyen arányú pontatlanság.

Mindezek a megfontolások azt bizonyítják, hogy a lineáris egyenletrendszer megoldásának feladata az ún. *inkorrektül felállított feladatok* közé tartozik. Példánkban egy ún. *rosszul kondicionált mátrix* szerepelt: ilyenkor nem feltétlenül értelmes dolog az egyenlet pontos megoldásáról beszélni. Sok érdekes elméleti problémát vet fel a közelítő megoldás olyan értelmezése, amely a fenti hibáktól mentes. Új feladatot jelent a jó algoritmusok keresése közelítő megoldások kiszámítására. Ezzel a kérdéssel kiterjedt irodalom foglalkozik. Mi most csak azt mutatjuk meg (*Borosh és Fraenkel* nyomán), hogy ha a bemenő adatok hibáin már nem is áll módunkban változtatni, a kerekítési hibák nélküli számolás gyakorlatilag megvalósítható. Ehhez a 3.5.1. pontban ismertetett számelméleti fogásokat használjuk fel.

A feladat bemenő adatait, az a_{ij} és a b_i számokat egészeknek tekinthetjük. (Ha nem lennének azok, közös nevezőjükkel beszorozva egészzé tehetők.) Az eljárás a következő ismert algebrai tényt használja fel.

- **Tétel (Cramer-szabály):** Jelöljük $u_1, u_2, \dots, u_n$ -nel a következő determinánsok értékeit:

$$u_1 = \begin{vmatrix} b_1 & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ b_n & a_{n2} & \dots & a_{nn} \end{vmatrix}, \quad u_2 = \begin{vmatrix} a_{11} & b_1 & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & b_n & \dots & a_{nn} \end{vmatrix}, \quad \dots$$

$$\dots, u_n = \begin{vmatrix} a_{11} & \dots & a_{1(n-1)} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \dots & a_{n(n-1)} & b_n \end{vmatrix}.$$

Jelöljük K -val az A mátrix determinánsát:

$$K = \begin{vmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{vmatrix}.$$

Ekkor az $u_1, \dots, u_n$ egész számok kielégítik az $Au = Kb$, azaz az

$$a_{11}u_1 + \dots + a_{1n}u_n = Kb_1$$

$$\vdots$$

$$a_{n1}u_1 + \dots + a_{nn}u_n = Kb_n$$

egyenletrendszert. Ha $K \neq 0$, akkor más megoldás nincs is.

Tegyük fel most az egyszerűség kedvéért, hogy $K \neq 0$. A moduláris algoritmusok alkalmazása közben érteni kell ahhoz, hogy egy determináns abszolút értékét egyszerű módon felülről megbecsüljük. Erre alkalmas eszköz *Hadamard* tétele:

• **Tétel (Hadamard):**

$$K \leq (a_{11}^2 + \dots + a_{n1}^2)^{1/2} \dots (a_{1n}^2 + \dots + a_{nn}^2)^{1/2}.$$

Ha pl. tudjuk, hogy az a_{ij} számok mindegyike kisebb, mint M , akkor innen azt kapjuk, hogy

$$K \leq (\sqrt{n}M)^n.$$

Tipikus nagyságrendek: $M = 10^7$ (egy rekeszben található legnagyobb egész szám), $n = 100$. Ekkor tehát $K \leq 10^{800}$.

Következmény:

Ha $a_{ij} \leq M$ és $b_i \leq M$ ($i, j = 1, \dots, n$), akkor $u_i \leq (\sqrt{n}M)^n$.

Mivel a (4) egyenletrendszer megoldásai az

$$\frac{u_1}{K}, \dots, \frac{u_n}{K}$$

törtszámok, feladatunknak ezután a $K, u_1, \dots, u_n$ egész számok meghatározását tekintjük. A megoldások lépései:

1. Szükségünk van egy olyan prímszámtáblázatra, amely sok, M -nél kisebb, de ezen belül minél nagyobb prímszámot tartalmaz. Kiválasztunk egy ilyen p_1 prímszámot. Kiszámoljuk K maradékát mod p_1 . Ennek egy lehetséges módját a következő példa mutatja.

Példa:

$$D = \begin{vmatrix} 850 & -73 & 1024 \\ 1 & 68 & 678 \\ 5 & -98 & 136 \end{vmatrix} \text{ maradékának kiszámítása mod 11.}$$

Első lépésként az egyes számokat redukáljuk mod 11, vagyis a legkisebb maradékukkal helyettesítjük őket. (Ezt a redukciót minden közbenső lépés után megismételjük.) Ezután a háromszög alakra hozás (a főátló fölött csupa 0 lesz) módszerét követjük, mod 11. Cseréljük fel az első oszlopot az utolsóval:

$$D \equiv \begin{vmatrix} 3 & 4 & 1 \\ 1 & 2 & 7 \\ 5 & 1 & 4 \end{vmatrix} \equiv - \begin{vmatrix} 1 & 4 & 3 \\ 7 & 2 & 1 \\ 4 & 1 & 5 \end{vmatrix} \pmod{11}.$$

Vonjuk le az első oszlop 4-, ill. 3-szorosát a második, ill. a harmadik oszlopból:

$$D \equiv - \begin{vmatrix} 1 & 0 & 0 \\ 7 & 2-28 & 1-21 \\ 4 & 1-16 & 5-12 \end{vmatrix} \equiv - \begin{vmatrix} 1 & 0 & 0 \\ 7 & 7 & 2 \\ 4 & 7 & 4 \end{vmatrix} \pmod{11}.$$

Emeljük ki a második oszlopból 7-et, majd vonjuk le a második oszlop kétszeresét a harmadikból:

$$D \equiv -7 \cdot \begin{vmatrix} 1 & 0 & 0 \\ 7 & 1 & 2 \\ 4 & 1 & 4 \end{vmatrix} \equiv -7 \cdot \begin{vmatrix} 1 & 0 & 0 \\ 7 & 1 & 0 \\ 4 & 1 & 2 \end{vmatrix} \pmod{11}.$$

A háromszög alakú determináns értéke egyenlő a főátlóban álló elemek szorzatával:

$$D \equiv -7 \cdot 2 \equiv 8 \pmod{11}.$$

Miután kiszámoltuk K maradékát mod p_1 , nézzük meg, nem nulla-e ez a szám? Ha igen, akkor dobjuk el p_1 -et, mert számunkra nem használható: p_1 ún. *szerencsétlen prímszám*. Próbálkozzunk egy újabb, p_2 prímszámmal. Annak, hogy valaha is szerencsétlen prímszámmal találkozzunk, igen kicsi a valószínűsége. Mint tudjuk, $K \neq 0$. A szerencsétlen prímszámok éppen azok, amelyek osztói K -nak. Márpedig p_1 legfeljebb $1/p_1$ valószínűséggel lesz osztója K -nak. (Emlékezzünk arra is, hogy p_1 igen nagy szám, közel akkora, mint M .)

Jelölje $K^{(1)}$ a K legkisebb maradékát mod p_1 . Ha $K^{(1)} \neq 0$, akkor p_1 megfelel. Vegyünk ekkor egy p_2 prímszámot és ismételjük

meg az előbbi eljárást. Ezt folytassuk addig, amíg végül rendelkezésünkre állnak a

$$p_1, p_2, \dots, p_s$$

prím számok, melyekre mind kiszámoltuk K maradékának, $K^{(1)}, \dots, K^{(s)}$ -nek az értékét, és amelyekre

(I) $K^{(1)}, \dots, K^{(s)}$ egyike sem 0,

$$(II) p_1 p_2 \dots p_n > (\sqrt{n} M)^n.$$

(Ha $n=100$, $M=10^7$, a prím számok pedig közel M nagyságúak, akkor s értéke kb. 120 lesz.)

2. Oldjuk meg az

$$Ax \equiv K^{(1)}b \pmod{p_1},$$

kiírva

$$\begin{aligned} a_{11}x_1 + \dots + a_{1n}x_n &\equiv K^{(1)}b_1 \\ &\vdots \\ a_{n1}x_1 + \dots + a_{nn}x_n &\equiv K^{(1)}b_n \end{aligned} \pmod{p_1} \quad (5)$$

kongruenciarendszert. Mivel $K \not\equiv 0 \pmod{p_1}$, a megoldás egyértelmű lesz.

Példa:

$$\begin{aligned} 850x_1 - 73x_2 &\equiv 1024 \\ 5x_1 - 98x_2 &\equiv 136 \end{aligned} \pmod{11}$$

kongruenciarendszer megoldása. Redukáljuk mod 11:

$$\begin{aligned} 3x_1 + 4x_2 &\equiv 1 \\ 5x_1 + x_2 &\equiv 4, \text{ azaz } x_2 \equiv 4 - 5x_1 \pmod{11}. \end{aligned}$$

Behelyettesítve:

$$3x_1 + 4(4 - 5x_1) \equiv 5x_1 + 5 \equiv 1, \quad \text{azaz} \quad 5x_1 \equiv -4 \equiv 7 \pmod{11}.$$

Szorozzunk be 5 inverzével, 9-cel:

$$x_1 \equiv 7 \cdot 9 \equiv 8, \quad x_2 \equiv 4 - 5 \cdot 8 \equiv 8 \pmod{11}.$$

A kongruenciarendszert tehát úgy oldjuk meg, mint az egyenletrendszert szokás, csak

- minden lépés után redukálunk a legkisebb maradékra, és
- osztás helyett „reciprokkal” való szorzást végzünk, ahol a reciprokot a 3.5.1. pontban ismertetett módon, az euklidészi algoritmussal keressük meg.

Ha megvan az (5) kongruenciarendszer $u_1^{(1)}, \dots, u_n^{(1)}$ megoldása, akkor oldjuk meg hasonlóan a $p_2, \dots, p_s$ prímszámokra is az (5)-nek megfelelő kongruenciarendszert. Végül tehát rendelkezésünkre fognak állni az

$$\begin{aligned} \mathbf{u}^{(1)} &= (u_1^{(1)}, \dots, u_n^{(1)}), \\ &\vdots \\ \mathbf{u}^{(s)} &= (u_1^{(s)}, \dots, u_n^{(s)}) \end{aligned}$$

megoldások.

3. A kínai maradéktétel (ill. az utána leírt eljárás) segítségével számítsuk ki azokat a legkisebb abszolút értékű $L, v_1, \dots, v_n$ számokat, amelyekre

$$\begin{aligned} L &\equiv K^{(1)}, \quad v_1 \equiv u_1^{(1)}, \dots, v_n \equiv u_n^{(1)} \pmod{p_1}, \\ &\vdots \\ L &\equiv K^{(s)}, \quad v_1 \equiv u_1^{(s)}, \dots, v_n \equiv u_n^{(s)} \pmod{p_s}. \end{aligned}$$

Egyszerűen belátható, hogy $L=K, v_1=u_1, \dots, v_n=u_n$, tehát meghatároztuk a keresett számokat.

Azzal az esettel, amikor a mátrix K determinánsa nulla is lehet, nem foglalkozunk. Ez lényegében a fentihez hasonló módon intézhető el.

Összefoglalva: Ha egy 100×100 -as méretű egyenletrendszert akarnánk kerekítés nélkül megoldani, akkor menet közben esetleg több száz jegyű számokkal kellene dolgozni. A bemutatott módszer lehetővé teszi, hogy ehelyett mintegy 120 darab kongruenciarendszert oldjunk meg, végig közönséges (single precision) számokkal dolgozva.

Érdekes azt is megjegyezni, hogy a 100 darab moduláris számolás elvben egyidőben is végezhető lenne, különböző számítógépeken. Az egyetlen akadályt a „szerencsétlen prímek” jelentik, de ezek igen ritkák. Ha a megoldás nem egyértelmű, a párhuzamosításra más módszerek vannak, melyekkel itt nem foglalkozunk.

4. fejezet

A kiszámíthatóság elméletéről

4.1.

Az algoritmikus megoldhatóság

Vannak-e olyan feladatok, amelyekhez nem is lehet megoldó algoritmust találni? Ez a kérdés a gyakorlati ember számára erősen spekulatívnak tűnik: ő konkrét algoritmusokat keres a rá háruló konkrét feladatok megoldására. Jobban meggyőzhetünk valakit a kérdés érdekességéről, ha olyan feladat megoldhatatlanságát bizonyítjuk be, amivel már ő is próbálkozott. Nos, a matematika bizonyos ágaiban (különösen az automaták és formális nyelvek elméletében) valóban lépten-nyomon eldönthetetlen feladatokba ütközünk. Ezért egy negatív eredmény is lehet hasznos: ahogy senki nem foglalkozik többet a kör négyszögesítésével, mióta a múlt században megmutatták, hogy a π szám transzcendens, úgy senki sem keres eljárást annak eldöntésére, hogy adott elemi (az alapműveletek, a hatványozás, a szinusz-, koszinusz- és a logaritmusfüggvények segítségével kifejezhető) függvény integrálja elemi függvény-e, mióta tudjuk, hogy ilyen nincs.

Csak akkor mutathatjuk meg, hogy egyes feladatoknak nincs megoldó algoritmus, ha előbb definiálni tudjuk, milyenek van? Az algoritmus fogalmának definíciójára más célból is szükség lehet. Vannak ugyanis olyan algoritmusok, amelyek működésük közben az összes lehetséges algoritmust valamilyen formában figyelembe veszik. A jó sakkozó például az egyes szituációkban nem egyszerűen az optimális lépéseket lépi (ehhez csillagászati méretű számolásra lenne szüksége), hanem próbálja kitalálni az ellenfél stratégiáját: felkészül azok ellen a stratégiák ellen, amelyek az ellenfelet feltehetőleg vezérlik. A stratégia nem más, mint algoritmus. Algoritmus végtelen sok van, de a sakkozó csak néhány legegyszerűbbet képes figyelembe venni. Ehhez viszont már azt is tudni kell, milyen algoritmust tekintünk egyszerűbbnek, milyent bonyolultabbnak?

4.1.1.

Az algoritmus fogalma

Az algoritmus fogalmának nincsen egységes, elfogadott matematikai definíciója. Az utóbbi negyven évben sok különböző matematikai modell született, amiben algoritmusokat lehet megfogalmazni. Ezeknek a modelleknek összevetése arra a felismerésre vezetett, hogy noha az egyes modellekben egészen mást értenek algoritmuson, *a feladatok köre*, amelyeket algoritmusokkal meg lehet oldani, mindig ugyanaz.

Szorítkozzunk azokra az esetekre, amikor a feladat egy $f(x)$ függvény kiszámítása.

Milyen függvények jönnek szóba? Ha a függvény értelmezési tartományában vagy értékkészletében valós számokat is megengednénk, akkor már nem volna egyszerű dolog kiszámolhatóságról beszélni. Egy algoritmus csak jól megfogható, véges objektumokkal operál. Mármint a véges objektumokat (egész számokat, törteket, sőt olyan absztrakt objektumokat is, mint gráfok, az ábécé betűiből alkotott szavak, formulák stb.) könnyű reprezentálni néhány természetes számmal. Ezért egyezzünk meg abban, hogy $f(x)$ függvényünk értelmezési tartománya is, értékkészlete is *a természetes számok véges sorozataiból* fog állni.

$$x = (x_1, x_2, \dots, x_k), \quad f(x) = y = (y_1, \dots, y_e).$$

Feladatnak tekintjük tehát például az olyan $f(x)$ függvény kiszámítását, amelynek értelmezési tartománya (bemenő adatai) és értékkészlete (kimenő adatai) is természetes számok.

4.1. feladat: Adjunk algoritmust a következő függvény kiszámítására:

$$f(x) = \begin{cases} 0, & \text{ha } x \text{ prímszám} \\ 1 & \text{különben.} \end{cases}$$

Minden *tulajdonságnak* egy olyan függvényt feleltethetünk meg, amely 1 értéket vesz fel, ha a tulajdonság teljesül és 0 értéket különben. Ezért azt eldönteni, hogy az adott tulajdonság teljesül-e vagy nem, ugyanazt jelenti, mint a függvényt kiszámolni. Mint említettük, az algoritmus fogalmát sokféleképpen lehet értelmezni, de ez a sokféleség nem befolyásolja a feladatok algoritmikus megoldhatóságának kérdését. Kiderült, hogy az $f(x)$ függvény algoritmussal való kiszámolhatósága független attól, mit értünk formálisan algoritmuson. Természetesen szükséges, hogy a formális definíció megfeleljen az algoritmusról alkotott intuitív képnek.

Nem könnyű hirtelenjében olyan függvényt mutatni, amely nem számítható ki algoritmussal, illetve olyan tulajdonságot, amelynek teljesülését nem lehet algoritmussal eldönteni. Pedig bizonyos értelemben a függvények többsége ilyen: a fenti típusú függvények halmaza kontinuum, az algoritmusoké azonban csak megszámlálható számosságú.

Azokat a függvényeket, melyekhez van őket kiszámoló algoritmus, *kiszámítható (rekurzív) függvénynek*, azokat a tulajdonságokat, amelyek algoritmussal eldönthetők, *(rekurzíve) eldönthető tulajdonságoknak* nevezzük. Most megpróbálunk képet adni a kiszámítható függvényekről.

Legtöbb olvasónkról feltételezhető, hogy van intuitív fogalma, közvetett vagy közvetlen tapasztalata arról, hogyan működik egy számítógép. Sokféle számítógép van, melyek igen különböző módon (gépi kódban, különböző programnyelveken) programozhatók, belső felépítésük technikai megoldása is eltér egymástól. A működés alapvető elvei azonban oly mértékben közösek, hogy a programvezérlésű számítógépek — ha a tárhelykapacitástól eltekintünk — az általuk megoldható feladatok halmazát tekintve nem térnek el egymástól (természetesen a végrehajtás az egyiken sokszor gyorsabb, a programozás egyszerűbb lehet, mint a másikon).

Képzeljünk el most *Turing* nyomán egy ideális számítógépet, amely

1. soha nem hibázik,
2. akármennyi ideig képes dolgozni tönkremenés nélkül,
3. tárhelykapacitása korlátlanul bővíthető.

Mindhárom korlátozó tényező figyelmen kívül hagyása valóban erős idealizáció! Mindenesetre, ha egy feladatról kiderül, hogy megoldhatatlan az ideális gépen, akkor méginkább megoldhatatlan lesz a valódi gépen. Nyilván elképzelhető bármely mai digitális számítógép fenti „ideális” formája.

Algoritmuson ezentúl olyan eljárást értünk, amely egy ideális számítógépen programozható.

A konkrétság kedvéért most leírunk egy nagyon egyszerű T gépet, amely azonban lehetőségeiben bármely, ma használatos gépet felülmúl.

A gépnek két része van: a tár és a vezérlőegység. Utóbbi hajtja végre a programot.

A tár végtelen sok regiszterből áll, amelyek az 1, 2, 3, ... számokkal vannak számozva. Minden regiszter egy tetszőleges természetes számot tartalmazhat, de a gép működése folyamán mindig csak véges sok regiszter tartalma különbözik 0-tól. Az n -edik regiszter tartalmát $X[n]$ -nel jelöljük. (Éppen $X[n]$ tetszőleges nagysága teszi esetenként igen gyorsá ezt a számítógépet: az n -edik rekeszből közvetlenül át lehet ugrani az $X[n]$ -edikbe!) Ezenkívül van egy „indexregiszter”, mely egy I -vel jelölt tetszőleges természetes számot tartalmazhat.

A gépre írt program utasításokból áll. Háromfajta utasítás van: művelet-végrehajtó, feltételes ugró és a STOP. Egyes utasításoknak száma van, melyet az utasítás elé írunk.

A STOP-ra a gép megáll. A feltételes utasítás ilyen alakú:

IF $I = X[0]$ THEN p .

A gép ennek hatására ellenőrzi az $I = X[0]$ feltételt. Ha az teljesül, akkor áttér a p számú utasításra, ha nem, akkor a soron következőre (ha nincs p számú utasítás, vagy soron következő utasítás, akkor megáll).

A műveletvégrehajtó utasítás a következő hat fajtájú lehet:

$I \leftarrow 0$	$X[n] \leftarrow I$	$X[0] \leftarrow X[I]$
$I \leftarrow I + 1$	$I \leftarrow X[n]$	$X[I] \leftarrow X[0]$

Vegyük pl. az $X[0] \leftarrow X[I]$ utasítást. Ennek hatására a gép a 0-dik rekeszbe beírja az I -edik rekesz tartalmát, ahol I az indexregiszter tartalma.

Hogyan lehet a gépen egy $f(x)$ függvényt kiszámolni? A függvény $x = x_1, x_2, \dots$ bemenő adatait sorba beírjuk az $X[1], X[4], X[7], \dots$ helyekre. Beindítunk egy P programot. Ha az megáll, akkor az $y = f(x)$ eredményt kiolvashatjuk az $X[2], X[5], X[8], \dots$ helyekről. (Az $I, X[0], X[3], X[6], \dots$ helyeket a gép számára szabadon hagyjuk.)

Az $f(x)$ függvényt a T gépen kiszámíthatónak nevezzük, ha van olyan P program, amely ilyen módon kiszámítja. Ez már szigorú matematikai definíció, amit más formában Church fogalmazott meg, az 1930-as években.

A híres Church-tézis így hangzik: Minden függvény, amely intuitív értelemben kiszámítható, a T gépen is kiszámítható lesz.

A Church-tézist igen sok tapasztalati és elméleti meggondolás támasztja alá, ezért általánosan elfogadott.

4.1.2.

Az univerzális gép. A megállási feladat

Említettük, hogy ha egy ma használatban levő programvezérlésű számítógép idealizált változatát vesszük, akkor ezen a gépen (nevezzük T-nek), minden elképzelhető algoritmus beprogramozható. Az ilyen gépet e tulajdonsága miatt univerzálisnak nevezzük. A T gép univerzalitása azt jelenti, hogy T-n akármilyen más T idealizált számítógép működése — alkalmas programmal — szimulálható.

A T számítógép működését közelebbről a következőképpen képzeljük el. Van három nyílás. Az elsőn a programot, a másodikon

az adatszalatot lehet beadni. A bemenő adatok az adatszalatagon egy $x = (x_1, x_2, \dots, x_k)$ számsorozattal vannak felírva. A P program is kódolható egy 0-kból és 1-esekből álló sorozattal. Jelöljük ezt a kódot $\bar{P}$ -vel: $\bar{P} = (p_1, p_2, \dots, p_l)$. A program és az adatok beadása után a gép működésbe lép. Ha a program lefutott, akkor a gép a harmadik nyíláson kiadja az eredményszalatot, melyre az eredmény van nyomtatva, $y = (y_1, \dots, y_n)$ számsorozat formájában. Ezt az eredményt $y = T(P, x)$ -szel jelöljük.

A T gép univerzalitása formálisan a következőt jelentheti: Minden más S géphez található a T gépen egy olyan P_s program, hogy akárhogy adjunk meg az S géphez egy Q programot és x bemenő adatokat, a T gép a P_s programmal ugyanazt számolja ki a $\bar{Q} + x$ számsorozatból, mint az S gép az x -ből. Formulával: $T(P_s, \bar{Q} + x) = S(Q, x)$ (a $+$ jelet $\bar{Q}$ és x közé csak elválasztás céljából írtuk).

Most mutatunk egy olyan feladatot, amelynek megoldására nincs algoritmus. Ez az ún. *megállási feladat*.

Döntsük el, hogy számítógépünk adott P program és x adatszalat mellett a végtelenségig dolgozik-e vagy véges időn belül megáll?

(Megjegyezzük, hogy ez a feladat nagyon is természetes: jó volna a programokról legalább annyit eldönteni, hogy nem futnak-e végtelen ideig. Nyilván nem jó eljárás kipróbálni P -t és x -et: ha a program végtelenségig dolgozik, közben nem tudhatjuk, meg fog-e állni).

4.2. feladat: Adjunk algoritmust annak eldöntésére, hogy egy gép adott P program és x adatok mellett végtelen ciklusba kerül-e?

4.3. feladat: A megállási feladatban egy programról és egy számsorozatról kell eldönteni valamit. Fogalmazzuk át a feladatot úgy, hogy csak számsorozatról legyen benne szó!

Tegyük fel, hogy a kérdés eldöntésére tudnánk algoritmust találni. Ez akkor speciálisan abban az esetben is működne, ha $x = \bar{P}$; más szóval, eldönthető volna az alábbi kérdés is:

(*) Adott a P program: döntsük el, hogy ha az adatszalatra $\bar{P}$ -t és a programszalatra P -t írunk, véges időn belül megáll-e a gép?

Gépünk univerzális. Ezért, ha ilyen algoritmus létezik, akkor T számára írható olyan Q program, amely bármely P program esetén eldönti, hogy az adatszalatra $\bar{P}$ -t és a programszalatra P -t írva, végtelen ideig dolgozna-e a gép. Ehhez persze a Q -val dolgozó gép számára az adatszalatra $\bar{P}$ -t kell írni: a gép $T(Q, \bar{P})$ -vel jelzi az eredményt.

Hogyan jelezze ezt? A bizonyítás szempontjából a következő célszerű programozni (noha ezt nyilván nem tennénk a gyakorlatban): Álljon le a gép, ha a válasz igenlő, vagyis az adatszalagra $\bar{P}$ -t, a programszalagra P -t írva a gép végtelen ideig dolgozna! Fusson a gép a végtelenségig, ha az derül ki, hogy a P program a $\bar{P}$ adatszalagon csak véges ideig dolgozna (itt van a furfang a dologban!).

Nézzük most a következőt: mi történik, ha az adatszalagra $\bar{Q}$ -t, a programszalagra Q -t írunk?

Ha a gép véges időn belül leáll, akkor, mint tudjuk, ez azt jelzi, hogy Q önmagán végtelen sokáig dolgozik. De nem dolgozhat végtelenségig, hiszen éppen azt tettük fel, hogy véges időn belül leáll!

Ha a gép végtelenségig dolgozik, ugyanúgy ellentmondásra jutunk. Ez ugyanis azt jelzi, hogy Q önmagán csak véges ideig dolgozik. Pedig az ellenkezőjét tettük fel!

Mindenképpen ellentmondásra jutottunk tehát. Nem lehetett helyes az a feltevésünk, hogy a megállási probléma algoritmikusan eldönthető, tehát nem dönthető el.

A fenti eldönthetetlenségi kérdésről megjegyezzük, hogy nem mesterkéltén felállított problémásereg, hanem természetesen felvetődhet. Ennek ellenére érezhetően (és a bizonyítást elolvasva lényegesnek tűnően) a buktató abban van, hogy a programokról akarunk program segítségével valamit kideríteni.

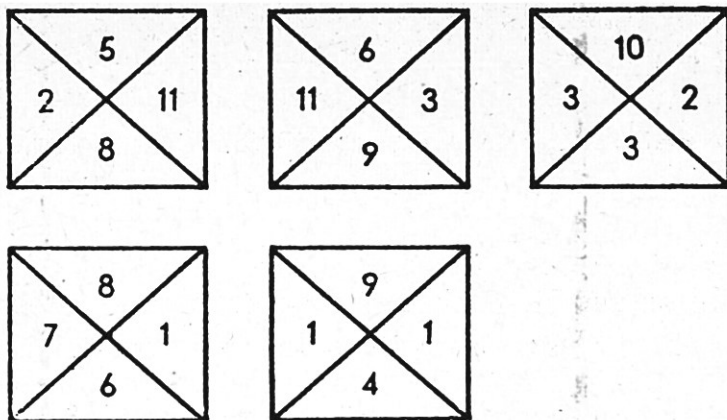
Valószínűleg senki sem tett még komoly erőfeszítést, hogy a megállási feladat megoldására algoritmust találjon. Talán csak úgy áll a helyzet, hogy vannak ilyen nyakatekert és szemmel láthatóan megoldhatatlan feladatok, de ezeket józan ésszel könnyen el lehet különíteni azoktól, melyeknek megoldására van remény? Nos, a dolog nem ilyen egyszerű. Az a probléma ugyanis, hogy adott feladat algoritmikusan megoldható-e vagy sem, ugyancsak az eldönthetetlenek sorába tartozik!

És vannak olyan feladatok, amelyek algoritmikusan megoldhatatlanok, de semmi közük a programokhoz.

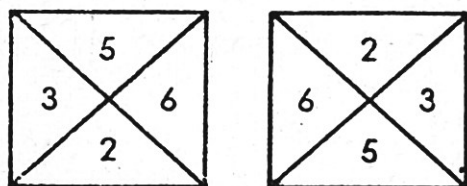
4.1.3.

Egyszerű eldönthetetlen feladatok

Az algebrában, topológiában és főleg a matematikai logikában sok olyan feladat vetődik fel, amely később eldönthetetlennek bizonyul. A matematikai gépek és nyelvek elméletében már — kis túlzással szólva — ahhoz kell ügyesség, hogy az ember az eldönthetetlen feladatok tömegéből kihalásson egy-egy el-



20. ábra



21. ábra

dönthetőt. De mint látni fogjuk, a számelmélet és az analízis nevezetes feladatai között is vannak eldönthetetlenek. Ezek közül három problémát mutatunk be.

Lássunk először egy játékos, kombinatorikai jellegű feladatot, a **d o m i n ó p r o b l é m á t**.

A dominók — mint tudjuk — téglalap alakú kövek, amelyeknek két végén számok vannak. Lerakáskor az azonos jelű végeknek kell egymás mellé kerülni. Mi egy fokkal nehezebb dominójátékot vezettünk be. A kövek négyzet alakúak lesznek (l. 20. ábra). Egy kőnek mind a négy oldalára az ábrán látható módon valamilyen szám van írva. A köveket úgy kell egymás mellé rakni, hogy a szomszédos oldalak jele azonos legyen (mint az ábrán). Az ilyen lerakást szabályosnak mondjuk.

A kövek felső, alsó, bal és jobb oldala rögzített, forgatni nem szabad őket. Ilyen módon minden kő négy számmal jellemezhető: (felső, alsó, bal, jobb). Például az ábrabeli bal felső dominó (5, 8, 2, 11) típusú. A feladat a következő: Tegyük fel, hogy kaptunk egy K készletet. A készletben véges sok, adott dominótípus van, például az az öt típus, ami az ábrán látható. Minden kapott típusból végtelen sok darab egyforma kövünk van.

Le tudjuk-e fedni az egész síkot szabályosan, az adott K készletből?

Nyilván meg lehet úgy adni a készletbeli típusokat, hogy a lefedés végrehajtható legyen. Pl. ha a 21. ábrán látható típusaink vannak, ezeket sakktáblaszerűen váltogatva, könnyű az egész

síkot lefedni, míg ha az egyiken egy számot elrontunk, akkor látható, hogy már nem boldogulunk. Felvetődik a kérdés, van-e olyan algoritmus, amely bármely K készlet esetén eldönti, hogy a lerakási feladat megoldható-e? A válasz negatív: ez eldönthetetlen probléma!

4.4. feladat: Mutassuk meg, hogy ha adott K készletből akármilyen nagy négyzetet szabályosan le lehet fedni, akkor az egész sík is lefedhető!

4.5. feladat: Mutassuk meg, hogy van olyan készlet, amelyből az egész sík lefedhető, de nem fedhető le periodikusan, vagyis úgy, hogy a síkot „nagy” négyzetekre osztjuk, és minden ilyen négyzetet ugyanúgy fedünk le.

A következő példánk Hilbert tizedik problémája. Hilbert 1900-ban még azon a véleményen volt, hogy minden matematikai probléma megoldható, ha eléggé állhatatosan és sokáig keresik a megoldást. Felállított azonban 23 olyan problémát, amelyek az akkori matematika eszközeivel kezelhetetlennek tűntek, és meg volt győződve arról, hogy többségükkel ebben az évszázadban még nem boldogul a tudomány. A problémák közül szám szerint a tizedik a következő volt:

Diofantoszi egyenletnek nevezzük a többváltozós, egész együtthatós algebrai egyenleteket, tehát pl. egy

$$3x^3 - x^2yz + 6z + 11 = 0$$

alakú egyenletet, ha bennünket csak az egész megoldások érdekelnek. Speciális diofantoszi egyenletekkel kétezer év óta foglalkoznak a matematikusok. Pl. az

$$x^2 + y^2 = z^2$$

egyenlet megoldásai az ún. pithagorászi számhármassok (pl. 3, 4, 5 egy ilyen számhármass), az

$$x^3 + y^3 = z^3$$

egyenletnek pedig nincs 0-tól különböző egész megoldása. A speciális egyenletek minden esetben speciális megoldási módszereket követeltek, amelyek más egyenleteknél csődöt mondtak. Hilbert azt a feladatot tűzte ki, hogy találjunk olyan általános módszert (algoritmust), amely minden egész együtthatós algebrai egyenlet esetén eldönti: van-e annak egész megoldása. Nem gondolt arra, hogy esetleg ilyen módszer nem létezik.

A Hilbert-problémák megoldása a század legnagyobbjaitól sok éven át tartó erőfeszítéseket követelt. Ezek nyomán több új matematikai ág jött létre. A huszadik századi matematika dinamizmusát szépen szemléltetik azok a kérdések, amelyekben Hilbert nem jól látott előre. Először: nem minden matematikai probléma

dönthető el. Ez *Gödel* munkája nyomán vált ismertté. Másodszor: *Hilbert* problémáit majdnem egytől egyig megoldották e században, ha ez néhány évtizedbe tellett is. Harmadszor: a tizedik probléma negatíve oldódott meg: 1970-ben az akkor 23 éves *Matyijaszevics* bebizonyította, hogy nincs algoritmus a diofantoszi egyenletek megoldhatóságának eldöntésére. (*Matyijaszevics* felhasználhatta *Davis*, *Robinson* és *Myhill* sokéves munkáját, amelynek eredményeképp már „csak” egy tiszta számelméleti feladat megoldása volt hátra). Ennek a — most negatív formában kimondott — eredménynek sok fontos konstruktív matematikai következménye is van, amelyekre nem térhetünk ki.

Hilbert tizedik problémája a számelmélet körébe tartozik. A klasszikus analízis területén se lehetünk azonban biztonságban az eldönthetetlen problémáktól. Tekintsük azt a kérdést, van-e egy egyenletnek valós gyöke. A vizsgált egyenletek $F(x)=0$ alakúak, ahol $F(x)$ a racionális számokból, az x változóból és a $\sin 2\pi x$ függvényből összeadás és szorzás segítségével felépített kifejezés. Pl. az

$$1 + 40x^2 - \sin^2(2\pi x) = 0$$

egyenlet nem oldható meg, az

$$x^3 + \sin\left[2\pi\left(x^2 + \frac{1}{2}\right)\right] = 0$$

egyenlet pedig igen. (Legyen $x=0$.)

Matyijaszevics eredményéből *Richardson* és *Wang* levezették, hogy nincs olyan algoritmus, mellyel bármely ilyen alakú egyenletről eldönthető lenne, hogy megoldható-e.

A levezetés úgy történik, hogy minden $P(x_1, \dots, x_n)=0$ diofantoszi egyenletnek megfeleltetünk egy $P(x)=0$ egyenletet. Ez utóbbi egyenletnek akkor és csak akkor lesz valós gyöke, ha a $P(x_1, \dots, x_n)=0$ egyenletnek van egész számokból álló megoldása. Ezért, ha a valós gyök problémája eldönthető volna, akkor a diofantoszi egyenletek problémáját is el lehetne dönteni. Ez utóbbi pedig, mint tudjuk, eldönthetetlen.

4.2.

A számolások terjedelméről

A kiszámíthatóságnak nemcsak elvi matematikai, hanem elvi fizikai határai is vannak. A kvantummechanika törvényei nem engedik se a műveleti sebesség korlátlan növelését, se a korlátlan miniaturizálást. Biztosak lehetünk benne, hogy a következő néhány

millió évben semmilyen gép nem lesz képes végigvizsgálni egyakár csak 200 elemű halmaz összes részalmazát; ezért aligha jogos dolog algoritmusnak nevezni azt az eljárást, amely ilyen utasítást tartalmaz.

Ahhoz, hogy közvetlenül alkalmazható eredményeket kapjunk, természetesen még egy lépcsőfokkal lejjebb kellene szállnunk — a gyakorlati kiszámolhatóság szintjére. Ezt a lépést nem célunk megtenni. Hiszen számítógép és számítógép óriási mértékben különbözhet egymástól.

Nemcsak arról van szó, hogy egyes gépek gyorsabbak (megtehetjük, hogy az algoritmus hosszát nem másodpercekben, hanem lépésszámban mérjük), hanem arról is, hogy az egyes lépéseket nem arányos idő alatt képesek a gépek megtenni; van olyan lépés, amely — a hardware-től függően — az egyik gépnél közvetlenül, a másikon esetleg 10—20 közbülső lépésen keresztül valósítható meg. Elméleti vizsgálatokon célszerű külön foglalkozni az idő- és külön a tárigénnyel, de a gyakorlati számítógépeken ez a két dolog egymással szorosan, mégpedig az adott gép sajátosságainak megfelelően összefügg. (Központi tár, háttértárak, be- és kimeneti rendszer stb.) Ezenkívül a gyakorlati kiszámolhatóságról tett megállapítások a gépek fejlődésével, szerkezeti változásával gyorsan érvényüket veszítik.

A számítások terjedelmének mérésére ezért olyan mértékeket kell használni, amelyeknek mindenféle gép esetén értelme van, és az elméletnek elsősorban olyan kérdésekkel kell foglalkoznia, amelyek mindenfajta gép esetén egyforma élességgel vetődnek fel.

Az effektív vagy gyakorlati kiszámolhatóság problémájával, a gép korlátos kapacitásának elvi kérdéseivel nem mindenki kerül szembe, aki számítógépeket használ. Számítási terjedelem szempontjából a gépek felhasználási területeit hozzávetőlegesen így osztályozhatnánk (a különböző osztályokba tartozó feladatok természetesen sokszor keverve fordulnak elő):

Adatfeldolgozás

Ez a számítógépek legközvetlenebbül kínálkozó alkalmazási területe. Ma a legtöbb számítógépidőt erre használják, és a gépek térhódításának sebessége itt igen nagy.

Az adatfeldolgozásban nemigen merülnek fel kiszámolhatósági problémák. A műveletek egyszerűek: idő- és tárigényük az adatok mennyiségével mérhető. Az adatok mennyisége viszont horribilis lehet. A matematikai problémák leginkább az adatok elhelyezésének, átrendezésének visszakeresésének stb. optimális módjával kapcsolatosak.

Tudományos és műszaki számítások, szimuláció

Az egyenletek, közönséges vagy parciális differenciálegyenletek megoldása, integrálás, közelítő módszerek, rendszerint elég „ki-egyensúlyozottan” veszik igénybe a számítógépet. Legyen i a be-menő adatok mennyisége, j a kimenő adatoké, ε a közelítés pon-tossága, $n = \max(i, j, 1/\varepsilon)$. A tárhasználat ekkor általában cn és cn^2 , a számolási idő pedig cn és cn^4 között szokott mozogni.

A természeti, termelési stb. folyamatok *szimulációja* rendsze-rint hasonló igénybevételt jelent (a differenciálegyenletek megol-dása sokszor ugyancsak a szimulációhoz hasonló módszerrel tör-ténik), de vannak olyan szimulációs feladatok is, amelyek exponen-ciális terjedelmű számolást igényelnek. Ilyenek a több részecske mozgását számoló, vagy kristályrácsba helyezkedésükkor az együttes energiát minimalizáló szimulációs eljárások: a kapacitás-igény a részecskék számával exponenciálisan nő.

Operációkutatás

Ide soroljuk mindazokat az optimumkeresési feladatokat, amelyek nem oldhatók meg a klasszikus analízis módszereivel. Az operációkutatás középpontjában kétségtől a *lineáris progra-mozás* és ennek kiterjesztései állnak. A lineáris programozás ma legelterjedtebb algoritmusa, a szimplex módszer, elvben exponen-ciális ideig tartó kereséshez is vezethet. (Újabban *Kacsiján* és *Kar-markar* polinomiális idejű lineáris programozási algoritmusokat ad-tak.) Az összes gyakorlati esetben a lineáris programozás kapacitás-igénye ugyanúgy becsülhető, mint az előző részben: ha az isme-retlenek száma n , akkor a szükséges iterációk (egyenletrendszer-megoldások) száma cn . Gyakran merülnek fel olyan optimum-keresési feladatok, amelyeket eddigi módszereinkkel csak remény-telenül sok művelettel lehetne megoldani (pl. egészértékű programo-zás). Ilyenkor, ha nem sikerül hatékony algoritmust találni, meg-elégszünk gyorsabb, az optimumot úgy-ahogy megközelítő eljá-rással.

„Mesterséges intelligencia”

Itt olyan feladatokról van szó, amelyeket maga az ember jól-rosszul megold, de módszerét a gép számára egyelőre nem tudja programozni. E területen eddig kevés az egyértelmű siker.

Az *alakfelismerési*, *„tanulási”* és *gépi fordítási* feladatok az ember „beviteli” egységeinek munkájával analóg tevékenységet kívánnak a géptől. Már működnek ugyan egyszerű célokra (pl. kardiográfiai görbék elemzése) jól alkalmazható alakfelismerő algoritmusok, de bonyolultabb feladatok megoldásához fontos alapvető elvek hiányoznak. Az elvi kidolgozás nélkül, naivan írt

algoritmusok pedig megvalósíthatatlan mennyiségű *tárat* vennének igénybe (a gépi fordítás esetén pl. nyelvtani szerkezetek variánsainak tömegét, külön-külön fordítással együtt).

A *problémamegoldó algoritmusok*, pl. tételbizonyító programok, rendszerint egy általánosan fogalmazott keresési feladatba igyekeznek minél többet beépíteni az ember saját, heurisztikus keresési technikáiból. Meggyőző eredmények még e területen sem születtek. Elterjedt az a vélemény, hogy egy n hosszúságú bizonyítás megtalálásához a 2^n hosszúságú keresés általában elkerülhetetlen; de nem kizárt egyúttal, hogy azokat a fogásokat, amikkel ezt az emberi intelligencia gyorsítani tudja, a gépbe is be lehet programozni. A keresésre jellemző módon, a tárigény kicsi. (Amit elvettünk, nem kell feljegyezni.)

Láthatjuk, hogy a mai számítógépes gyakorlatban még elég ritka az olyan feladat, ahol az adatok mennyiségéhez mérten is nagyon nagy tömegű számolást kellene végezni. A jövőben az ilyen feladatok fontossága elkerülhetetlenül növekszik. Ezért az utóbbi másfél évtizedben nagy figyelmet szenteltek a számítások terjedelmével, bonyolultságával kapcsolatos elméleti kérdéseknek. Kiderült, hogy egészen egyszerűen hangzó problémák megközelíthetetlenek a rendelkezésre álló eszközökkel: igazi kihívást jelentenek a kor matematikájának! Ugyanakkor, olyan évszázadok óta meghitten ismerős feladatok, mint két szám szorzása, meglepetéseket rejtegetnek. Lassan olyan számítógépek jelennek meg, melyeknek gyorsmemória-kapacitása eléri az emberét. Ami ezekből a gépekből hiányzik: az intelligencia rejtélyes „programjai”. Ezeket megtalálni a tudomány számára nehéz, izgalmas és egyre gyakorlatibb feladat.

4.2.1.

A terjedelem mérése

Egy feladatra adott algoritmus esetén alapvető kérdés, *mennyi ideig tart* a számolás. Ez az általában igen nehéz kérdés különösen bonyolulttá akkor válik, ha azt is tudni akarjuk, hogy van-e gyorsabb (rövidebb idő alatt is célhoz érő) algoritmus.

A problémák ott kezdődnek, hogy az algoritmus hosszának fogalma sem egyértelmű. Két algoritmus közül egy adathalmazzal az egyik, más adathalmazzal a másik érhet előbb célhoz. Ezért többféle módon is lehet mérni az algoritmus időigényét. Szokás szerint a *legrosszabb eset* időigényét vesszük figyelembe: az algoritmust ekkor egy $f(n)$ függvény jellemzi, ahol $f(n)$ az a leghosszabb idő, amely n bemenő adat esetén az algoritmus lefuttatásához kell. Ennek hátránya az, hogy egy algoritmust leértékelünk csak azért,

mert annak néhány, esetleg igen mesterkéltén összeállított adathalmazon nagy az időigénye.

A legrosszabb eset időigényénél többet mond a gyakorlat számára az *átlagos időigény* (ismét rögzített számú bemenő adat esetén). Ez azonban matematikai nehézségeket rejt magában, és ritkán számítható ki. A legfőbb nehézség az, hogy a bemenő adatok eloszlása különböző gyakorlati területről jövő feladatok esetén más és más. Pl. ha valaki a folyamproblémát megoldó algoritmusokat közlekedési problémákban alkalmazza, olyan hálózatokat fog kapni, amelyek kereszteződés nélkül síkban ábrázolhatók. Ezzel szemben bonyolult csőhálózatok vizsgálata esetén már nincs ilyen leszűkítés. Az átlagos időigény vizsgálatának tehát csak akkor van értelme, ha a bemenő adatok eloszlása ismert, de kiszámítása még ez esetben is többnyire igen nehéz.

Újabban vizsgálták azt is, hogy mennyi az esetek legrosszabbikának időigénye, ha az *esetek kis százalékától eltekinthetünk*. Úgy is mondhatjuk, hogy az időigény „majdnem mindig” csak ennyi. Ez néha matematikailag jobban kezelhető, mint az átlagos időigény, másrészt megőrzi annak legfőbb előnyeit.

Az algoritmus időigényének kérdésében a fenti különbségtétel nemcsak elméleti, hanem a gyakorlatban használt fontos algoritmusoknál is mutatkozik. Ilyen példa a *szimplex-módszer*, a lineáris programozási feladatok megoldására. Victor Klee konstruált olyan példát, amelyben a szimplex-módszer az n^2 bemenő adaton 2^n lépésen keresztül dolgozik; így a legrosszabb esetben a szimplex-módszer időigénye exponenciálisan nagy. Ezzel szemben a szimplex-módszer átlagban — minden gyakorlati tapasztalat szerint — nem több, mint n^2 lépést igényel, sőt, az ennél több lépést igénylő bemenő adatok igen-igen ritkák. Így a tapasztalat azt mutatja (*Rényi* és mások bizonyos feltételek mellett bebizonyították ezt), hogy a szimplex-módszer időigénye átlagosan, sőt majdnem mindig n^2 .

Az idealizált számítógépen a számolás terjedelmének két leggyakoribb mértéke az *időigény* (lépésszám) és a *tárigény*. A gépek jelenlegi szerkezete mellett a gyakorlat számára érdekesebb volna az a szám, ahányszor a külső tárat kell igénybe venni, vagy ennél bonyolultabban meghatározható mértékszámok.

Elméleti vizsgálatokban egyelőre azonban a legegyszerűbb fogalmakra kell szorítkozni. Hiszen sokszor azt sem könnyű megbecsülni, hogy adott algoritmus mennyi ideig dolgozik! A lineáris programozás szimplex módszere például, mint említettük, bizonyos feladatokon 2^n lépést is tenne, ha hagynánk (n az ismeretlenek száma). Vannak azonban olyan speciális lineáris programozási feladatok, amelyekben, bizonyíthatóan, a lépésszám csak n^2 . Ilyen a szállítási feladat. Ennek bizonyítása azonban egyáltalán nem triviális (vö. 1.2.3. pont).

Másik példánk az *univerzális kereső feladatok* (mint például a pakolási feladat) megoldása. A 4.2.2. pontban vázolunk egy algoritmust, amely bizonyos értelemben optimális az e feladatot megoldó összes algoritmusok között. Erről az algoritmusról még az sem ismeretes, hogy a műveletek száma, amennyit a legrosszabb esetben végez, 2^n -nel, n^2 -tel vagy valami, a kettő közötti más $f(n)$ függvénnyel becsülhető-e?

Egyenletek megoldására és egyéb célokra gyakran használnak *iteratív közelítő módszereket*. A lépésszám itt attól függ, milyen gyorsan konvergálunk a megoldáshoz, és közismerten ezt sem mindig egyszerű megállapítani.

Az egyszerűbb algoritmusok többségénél mégis meg tudjuk becsülni a legrosszabb eset lépésszámát. Az átlagos lépésszám kiszámítása még ilyenkor is többnyire művészetet kíván.

A legnehezebb eset vizsgálatával a könyv sok algoritmusánál foglalkoztunk, egy algoritmusnál az átlagos lépésszámot is kiszámoltuk (l. 1.1.4. pont). Adott algoritmus esetén a számolás-igény (akár átlagban, akár a legrosszabb esetben) vizsgálatát az algoritmus *elemzésének* (analízisének) nevezzük.

4.2.2.

Optimális algoritmusok

Ha már sokszor azt is nehéz eldönteni, mennyi ideig dolgozik egy adott algoritmus, mennyivel nehezebb lehet megtalálni azt, amelyik a legrövidebb ideig dolgozik? Persze, ismét tisztázni kellene, mit veszünk figyelembe: az átlagot vagy a legrosszabb esetet? De még egy kérdés fölmerül: van-e legjobb algoritmus?

A feladat, amelyet meg kell oldani, legyen ilyen alakú: számítsunk ki egy adott $f(x)$ függvényt, ahol az x bemenő adathalmaz egy n hosszúságú bináris sorozattal $x = x_1, x_2, \dots, x_n$ ($x_i = 0$ vagy 1) van megadva, $f(x)$ értéke pedig valamilyen $y = y_1, y_2, \dots, y_m$ ($y_i = 0$ vagy 1) bináris sorozat.

Minden P programmal megadott algoritmushoz tartozik egy $t_P(x)$ függvény, amely megadja, hogy az adott algoritmus hány lépést dolgozik az x adathalmazon. Ha rögzítjük az adathalmaz n nagyságát, akkor kérdezhetjük, hány lépést végez az algoritmus a legrosszabb esetben? Ez a szám

$$t_P(n) = \max t_P(x),$$

ahol x végigfutja az összes n hosszúságú sorozatokat.

Ezek után megkérdezhetjük, van-e olyan P algoritmus, amely kiszámítja az $f(x)$ függvényt, és a legrosszabb esetbeli lépésszáma, $t_P(n)$, a lehető legkevesebb?

Minden adott n -re természetesen van ilyen P_n algoritmus. Egyáltalán nem biztos azonban, hogy van olyan algoritmus, ami egyszerre minden n -re optimális. Sőt vannak olyan feladatok, amelyeknek bármely P megoldó algoritmusánál van még egy sokkal jobb P' megoldó algoritmus is! Sokkal jobb abban az értelemben, hogy pl.

$$t_{P'}(n) \leq \log t_P(n)$$

minden elég nagy n -re.

Legtöbbször reménytelen vagy lehetetlen abszolút optimális algoritmust találni. Ezért üdvözölni kell minden olyan algoritmust is, amely csak valamilyen gyengébb értelemben optimális. Azt mondjuk, hogy a P algoritmus *konstans szorzó erejéig optimális*, ha bármely más, a feladatot megoldó Q algoritmus bármely x adathalmazon legfeljebb konstansszor dolgozik gyorsabban: azaz minden Q -ra van olyan $c_Q > 0$ állandó, hogy minden x -re

$$t_P(x) \leq c_Q t_Q(x).$$

Most egy elméleti szempontból igen fontos algoritmusról lesz szó: *Levinnek* a keresőfeladatokat megoldó, konstans szorzó erejéig optimális algoritmusáról.

A feladat a következő: adott egy n elemű G gráf, keressünk benne k darab független pontot. A feladat bemenő adatai tehát a G gráf és a k szám. Az 1.2.1. pontban láthattunk néhány algoritmust, melyek maximális számú független pontot találnak. Ezek az algoritmusok igen hosszú ideig dolgoztak, mert lényegében majdnem az összes részhalmazt végig kellett vizsgálniuk. Ha mostani feladatunkat így akarnánk megoldani, akkor $\binom{n}{k}$, tehát nagy k esetén ugyancsak sok lépésre lenne szükség: végig kellene vizsgálni az összes k elemű részhalmazt és megnézni nem lesz-e valamelyik független?

Persze azonnal leállhatunk, ha egy független k elemű részhalmazt találtunk. Ezért nem mindegy, milyen sorrendben nézzük végig a k pontból álló összes rendszert? Ne feledjük a célt: senki se tudjon nálunk ügyesebb lenni és gyorsabban találni egy k pontból álló független rendszert. Ha van olyan P programmal megadott algoritmus, amely gyorsan talál egy k elemű független rendszert, akkor mi sem akarunk lemaradni. Nos, ezt el lehet érni. Mindenekelőtt, az összes elképzelhető algoritmusokat P programjukkal, a P programot pedig kódjával, egy

$$\bar{P} = (p_1, p_2, \dots, p_l)$$

nullákból és egyesekből álló sorozattal lehet jellemezni.

Az összes 0—1 sorozatot pedig nyilván sorba lehet rakni (pl. úgy, ahogy a szótárban a szavakat) és végül is minden programhoz egyértelműen hozzárendelhetünk egy $m(P)$ számot, a P program számkódját. Mivel ez a hozzárendelés egyértelmű, minden m számhoz tartozik egy $P(m)$ program, mégpedig az, amit ő kódol:

$$P[m(P)] = P, \quad m[P(m)] = m.$$

Az optimális algoritmus a következőképpen dolgozik: számítógépünk a t -edik lépésben valamelyik P program működését szimulálja. Hogy melyiket, azt így állapítja meg: legyen m az a leggyobb szám, amelyre t osztható 2^m -mel. Ekkor a t -edik lépésben algoritmusunk a $P(m)$ program által megadott számolásban egyet továbblép. Ha a számolás éppen véget ért és kiadta a G gráf egy k elemű részalmazát, akkor algoritmusunk, mielőtt áttérnénk a $(t+1)$ -edik lépésre, ellenőrzi azt, hogy az adott k elemű részalmaz pontjai függetlenek-e? Ha igen, akkor leáll, ha nem, akkor továbblép, és a $(t+1)$ -edik lépésben egy más $P(m')$ program által megadott számolást folytat, ott ahol abbahagyta.

Ilyen módon a $P(m)$ program által megadott számolás a mi algoritmusunknak minden 2^{m+1} -edik lépésében szóhoz jut, és ezért, ha éppen ez a program találná meg gyorsan a független pontrendszert, akkor a mi programunk is, igaz, 2^{m+1} -szer lassabban, de megtalálja. Tehát ha a most leírt algoritmusnak a programja P_0 , akkor minden m -re és x -re:

$$t_{P_0}(x) \leq 2^{m+1} t_{P(m)}(x),$$

ezért algoritmusunk konstans szorzó erejéig optimális.

(Figyeljük meg, hogy noha a megadott algoritmusról tudjuk, hogy nála nincs jobb, sejtelmünk sincs róla, mennyi ideig is dolgozik valójában.)

4.6. feladat: Mutassuk meg, hogy van olyan c_0 szám, amellyel minden x -re és m -re

$$t_{P_0}(x) \leq c_0 m^2 t_{P(m)}(x).$$

4.2.3.

Alsó becslések

Algoritmusunk optimalitása rendszerint nem úgy derül ki, mint az előző pontban, hanem explicit módon. Kiderül pl., hogy az adott $f(x)$ függvény kiszámolása legrosszabb esetben legalább $\tau(n)$ lépésbe kerül: azaz minden P algoritmusra

$$t_P(n) \geq \tau(n); \tag{1}$$

másrészt megadható egy olyan algoritmus, amely $\tau(n)$ lépésben valóban kiszámítja az adott függvényt.

Azt írtuk, rendszerint így derül ki algoritmusok optimalitása, de az ilyen esetek is nagyon ritkák. Az (1) képletben a $\tau(n)$ függvény *alsó becslést* ad az $f(x)$ -et kiszámító algoritmusok lépésszámára. Ilyen alsó becsléseket egyelőre csak a legritkább és többnyire kevésbé érdekes esetekben sikerült kapni. Könyvünk kombinatorikai és algebrai részeiben nagyon erősen megkötöttük, milyen lépéseket engedünk meg az algoritmusban, hogy minél több algebrai és kombinatorikai technikát alkalmazhassunk. Így valóban lehet bizonyos típusú alsó becsléseket kapni, de csak nagyon szerényeket!

Nincs bebizonyítva, hogy két n jegyű számot nem lehet cn lépésben, két $n \times n$ -es mátrixot cn^2 lépésben, összeszorozni. Nem állíthatjuk biztosan, hogy nincs olyan algoritmus, amely n^2 lépésben megoldja a pakolási feladatot (ez is egy univerzális kereső feladat), pedig azt sőtjük, hogy minden megoldó algoritmus exponenciális mennyiségű lépést követel.

Van-e egyáltalán olyan feladat, amelyről tudjuk, hogy nagyon nehezen oldható meg? Igen! Hiszen láttuk, hogy vannak feladatok, melyeket egyáltalán nem lehet algoritmussal megoldani: ezek aztán igazán „nehezen” oldhatók meg! Ha pedig egy megoldhatatlan feladatot kissé módosítunk, elérhetjük, hogy megoldható legyen, de igen nehezen. Vegyük a megállási problémát: adott egy T számítógép és egy P program, eldöntendő, hogy a gép, ha elindítjuk az adott programmal, megáll-e valaha? Mint tudjuk, erre a problémára nincs általános eldöntő algoritmus. Módosíthatjuk a problémát a következőképpen: megáll-e a gép t darab lépés után? Várható, hogy ezt a problémát általános esetben nem lehet t -nél kevesebb lépésben megoldani. Az ide vonatkozó pontos matematikai állítások kimondására itt nincs módunk. Mindenesetre leszögezhetjük, hogy ebből az ún. korlátozott megállási problémából tetszőlegesen nehéz feladatok származtathatók. Ezek a feladatok nem mind gépekről és programokról szólnak, de általában automataelméleti vagy logikai jellegűek. Közülük nem egy már régebben ismert volt, ezért az, hogy nagyon megoldhatóságuk kiderült, fontos információ.

Feladatmegoldások

1. fejezet

1.1. Megmérünk 8 pénzdarabot, ebből megtudjuk, hogy köztük van-e a hamis, vagy a másik 8 között. Hasonlóan továbbmenve, a második méréssel 4-re, a harmadikkal 2-re, a negyedikkel 1-re tudjuk csökkenteni a hamisként szóba jövő pénzdarabok számát.

Három mérés nem elég, mert egy méréssel — kedvezőtlen esetben — legfeljebb a lehetőségek felét tudjuk kizárni.

1.2. Osszuk a 27 pénzdarabot három 9-es csoportba és kettőt hasonlítsunk össze. Ha valamelyikük könnyebb, abban van a hamis pénz; ha egyformák, akkor a harmadik csoportban. Hasonlóan továbbmenve, a második mérés 3-ra, a harmadik 1-re csökkenti a lehetőségek számát.

1.3. Mindkét eljárásban éppen azok az (x_i, y_j) párok kerülnek összehasonlításra, melyekre $x_{i-1} < y_j$ és $x_i > y_{j-1}$.

1.4. Egy elempárt legfeljebb egyszer kell *kicserélni*, és ezért legfeljebb $\binom{n}{2}$ csere kell. *Összehasonlítani* többször is szükséges lehet ugyanazt az elempárt, de azt észrevehetjük, hogy a sor első végigfutása után a legnagyobb elem már biztosan végső helyére (az utolsó helyre) kerül. Ezért a második végigfutás során már csak $n-2$ összehasonlítást kell végeznünk. Ugyanígy látható, hogy a harmadik, negyedik stb. végigfutásnál már csak $n-3$, $n-4$ stb. összehasonlítást kell végezni. Így az összehasonlítások száma összesen

$$(n-1) + (n-2) + \dots + 2 + 1 = \binom{n}{2}.$$

1.5. Legyen $\lceil \log_2 n \rceil = k+1$. Ekkor $n = 2^k + m$, ahol $1 \leq m \leq 2^k$. A bizonyítandó azonosság bal oldalát átalakítjuk:

$$\begin{aligned} & \underbrace{(k+1) + \dots + (k+1)}_{m\text{-szer}} + \underbrace{k + \dots + k}_{2^{k-1}\text{-szer}} + \underbrace{(k-1) + \dots + (k-1)}_{2^{k-2}\text{-szer}} + \dots \\ & \dots + \underbrace{2 + 2 + 1}_{2\text{-szer}} = \\ & = m(k+1) + k2^{k-1} + (k-1)2^{k-2} + \dots + 2 \cdot 2^1 + 1 \cdot 2^0 = \\ & = m(k+1) + (1 + 2 + \dots + 2^{k-1}) + (2 + 4 + \dots + 2^{k-1}) + \dots + \\ & + (2^{k-2} + 2^{k-1}) + 2^{k-1} = m(k+1) + (2^k - 1) + (2^k - 2) + \dots + \\ & + (2^k - 2^{k-1}) + (2^k - 2^k) = m(k+1) + (k+1)2^k - (2^{k+1} - 1) = \\ & = n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1. \end{aligned}$$

1.6. Teljes indukciót használunk. Legyen pl. $n = 2m$ (a páratlan m esete hasonlóan tárgyalható). Ekkor az első és második m elem sorbarakásához, az indukciós feltevés szerint,

$$2(m\lceil \log_2 m \rceil - 2^{\lceil \log_2 m \rceil} + 1)$$

összehasonlítás kell, majd az összefésüléshez még $2m-1$ lépés. Ez összesen

$$\begin{aligned} & 2m\lceil \log_2 m \rceil - 2^{\lceil \log_2 m \rceil} + 2 + 2m - 1 = \\ & = 2m\lceil \log_2 2m \rceil - 2^{\lceil \log_2 m \rceil} + 1. \end{aligned}$$

1.7. Minden játszmában egyetlenegy játékos esik ki.

1.8. Legyenek az elemek x_1, x_2, x_3, x_4, x_5 . Hasonlítsuk össze x_1 -et és x_2 -t! Az általánosság megszorítása nélkül feltehetjük, hogy az eredmény $x_1 < x_2$. Utána hasonlítsuk össze x_3 -at x_4 -gyel! Feltehetjük, hogy az eredmény $x_3 < x_4$. Hasonlítsuk össze x_1 -et x_3 -mal! Feltehetjük, hogy $x_1 < x_3$. Ekkor x_1 -ről már tudjuk, hogy kisebb, mint a mediáns, így a továbbiakban figyelmen kívül hagyhatjuk.

Maradt tehát 4 elem, melyek közül alulról a másodikat kell kiválasztani. Tudjuk már, hogy $x_3 < x_4$.

Hasonlítsuk össze x_2 -t és x_5 -öt! Legyen az eredmény mondjuk $x_2 < x_5$. Ezután hasonlítsuk össze x_2 -t és x_3 -at, és legyen mondjuk $x_2 < x_3$. Ekkor már biztosan tudjuk, hogy x_2 kisebb, x_4 pedig nagyobb, mint a keresett elem, így már csak x_3 és x_5 közül kell a kisebbiket kiválasztani, ami egy méréssel megtehető.

6-nál kevesebb összehasonlítás viszont nem elegendő. Tegyük föl ugyanis, hogy volna olyan eljárás, mely 5 összehasonlítással célhoz ér. Ez mindenképpen két elem összehasonlításával indul, és utána vagy két újabb elemet hasonlít össze egymással, vagy az első kettő valamelyikét egy újjal. Tegyük fel pl., hogy az első fajta eljárással van dolgunk; a második hasonlóan intézhető el. Ez első két lépésben így mondjuk $x_1 < x_2$ és $x_3 < x_1$ adódik. Az eljárásnak abban az esetben is működnie kell, amikor a tényleges sorrend $x_1 < x_3 < x_5 < x_4 < x_2$. Ekkor x_5 -öt még biztosan össze kell hasonlítani x_3 -mal és x_4 -gyel, mert sorrendjükre más mérésekből következtetni nem tudunk. De azt, hogy x_1 megelőzi x_5 -öt, csak onnan tudhatjuk, hogy x_1 -et még vagy x_3 -mal vagy x_5 -tel összehasonlítottuk; ugyanígy x_2 -vel is kellett, hogy végezzünk még egy összehasonlítást. Ez már 6 összehasonlítás.

1.9. Az ilyen gráfok diszjunkt utakból és sokszögekből állnak.

1.10. Ha S sem x -et, sem x szomszédjait nem tartalmazná, akkor x hozzávehető volna S -hez.

1.11. Ha S x -et nem tartalmazza, és x szomszédai közül is csak egy y -t, akkor y -t kihagyjuk és x -et betesszük S -be.

1.12. Ha a gráfban van legfeljebb elsőfokú pont, akkor az előző feladat szerint van olyan maximális független halmaz, mely ezt tartalmazza. Így a pontot és szomszédját elhagyva, elegendő a visszamaradó gráfban maximális független halmazt keresni.

Ha van másodfokú pont, akkor van olyan maximális független halmaz, amely vagy ezt a pontot, vagy mindkét szomszédját tartalmazza. Mindkét esetben egy-egy legfeljebb $n-3$ pontú gráfban kell maximális független halmazt találni, ami legfeljebb

$$2c \cdot 1,3^{n-3} = 0,91c \cdot 1,3^n < c \cdot 1,3^n$$

lépésben elvégezhető. Feltehetjük tehát, hogy minden pont foka legalább 3.

Tegyük fel most, hogy van egy harmadfokú pont, mondjuk x , és legyenek szomszédai u , v , w . Ha ezek háromszöget alkotnak, akkor a 2. feladatból az derül ki, hogy van x -et tartalmazó maximális független halmaz. Ekkor viszont egyetlen $n-4$ pontú gráf vizsgálatára redukálódik a kérdés. Feltehetjük tehát, hogy pl.

u és v nincsenek összekötve. A 2. feladatot felhasználva azt kapjuk, hogy van olyan maximális független halmaz, amely vagy x -et, vagy u -t, vagy v , w mindegyikét tartalmazza. Ezeket az eseteket egy-egy rendre legfeljebb $n-4$, $n-4$, $n-6$ pontú gráf vizsgálatára vezethetjük vissza, amihez a szükséges lépésszám csak

$$2c \cdot 1,3^{n-4} + c \cdot 1,3^{n-6} = 0,91c \cdot 1,3^n < c \cdot 1,3^n.$$

Ha van olyan pont, amely legalább ötödfokú, akkor az őt tartalmazó maximális független halmaz megkeresése egy $n-6$ pontú, az őt nem tartalmazóé egy $n-1$ pontú gráf vizsgálatát jelenti, amihez csak

$$c \cdot 1,3^{n-1} + c \cdot 1,3^{n-6} = 0,98c \cdot 1,3^n < c \cdot 1,3^n$$

lépés kell.

Így feltehetjük, hogy minden pont negyedfokú. Ha van két nem szomszédos pont, melyeknek legalább három szomszédja közös, akkor a 2. feladatból következik, hogy van olyan maximális független halmaz, mely vagy mindkettőt tartalmazza, vagy egyiket sem. Az első típusú független halmazok közül a maximális kiválasztása egy $n-6$ pontú, a második típusúak közül egy $n-2$ pontú gráf vizsgálatát jelenti. Ez elvégezhető

$$c \cdot 1,3^{n-6} + c \cdot 1,3^{n-2} = 0,8c \cdot 1,3^n < c \cdot 1,3^n$$

lépésben.

Így feltehetjük végezetül, hogy bármely két nem szomszédos pontnak legfeljebb két közös szomszédja van. Legyen x , y két szomszédos pont. Ekkor a 2. feladat eredménye szerint van olyan maximális független halmaz, mely vagy x -et, vagy y -t, vagy pedig x további három szomszédja közül (legalább) kettőt tartalmaz. Ez az öt eset rendre $n-5$, $n-5$, $n-8$, $n-8$, $n-8$ pontú gráf vizsgálatával elintézhető, és az ehhez szükséges lépésszám legfeljebb

$$2c \cdot 1,3^{n-5} + 3c \cdot 1,3^{n-8} = 0,91c \cdot 1,3^n < c \cdot 1,3^n.$$

Megjegyzendő, hogy a fenti esetek szétválasztása további lépéseket igényel, ezek azonban csak pontok fokainak, ill. szomszédainak meghatározásából állnak, és így cn^2 lépésben elvégezhetők. Ez pedig a fenti esetek mindegyikében még „fennmaradt” legalább $0,02 \cdot c \cdot 1,3^n$ lépéshez képest elenyészően kevés (ha n nagy).

1.13. Nyilvánvaló, ha van olyan pont, melyen k szállítás útja is keresztülmegy, akkor legalább k forduló kell. Másrészt, ha nincs olyan pont, melyen $k+1$ szállítás útja megy át, akkor k fordulóval meg is oldható a dolog a következőképpen: a teherautó elindul, az első A_i pontban megrakodik; ezt elviszi a megfelelő B_i pontba,

továbbmegy, és az első erre következő A_i pontban ismét meg-rakodik stb. Állítjuk, hogy ha az egész úton végigment, akkor nem lesz olyan pont, mely még mindig k elvégzendő szállítás útvonalán feküdne. (Ezzel kész is vagyunk, mert ezt k -szor ismételve, minden szállítás el lesz végezve.) Tegyük fel, hogy P ilyen pont. Mivel P eredetileg is csak k útvonalon lehetett rajta, P -n a teherautó üresen kellett hogy átmenjen. Legyen A_i a P előtti utolsó útvonal-kezdőpont. Ekkor A_i és P között útvonalvégpont nem lehet, mert egy ilyen és A_i közötti pont több, mint k útvonalon feküdne. Ezért a teherautó már A_i -ben üres kellett, hogy legyen, ami ellentmondás, mert akkor A_i -ben meg kellett volna rakodnia.

1.14. Nincsen.

1.15. Tegyük fel, hogy már néhány 1-est kijelöltünk a kívánt módon. Nevezzük ezeket kékeknek, a többit feketének. Húzzuk át az összes olyan oszlopot, melyből nem választottunk ki 1-est; majd amíg csak találunk olyan sort, mely az áthúzott oszlopok valamelyikét (fekete) 1-esben metszi, vagy olyan oszlopot, mely az áthúzott sorok valamelyikét kék 1-esben metszi, azt húzzuk át. Ha már nem tudunk mit áthúzni, két eset lehetséges:

a) Van olyan áthúzott sor, mely nem tartalmaz kék 1-est. Keressük meg azt a (fekete) 1-est, mely miatt ezt a sort áthúztuk, majd azt a kék 1-est, mely miatt az előbbi fekete 1-es oszlopát áthúztuk s. i. t. Végül visszajutunk egy olyan fekete 1-eshez, melynek oszlopában nincsen kék 1-es. Ekkor az így talált 1-esek színét megváltoztatva, az új kék 1-esek közül változatlanul nem lesz kettő egy sorban vagy oszlopban, és számuk nőtt.

b) Ha ilyen sor nincsen, a kék 1-esek száma máris maximális volt.

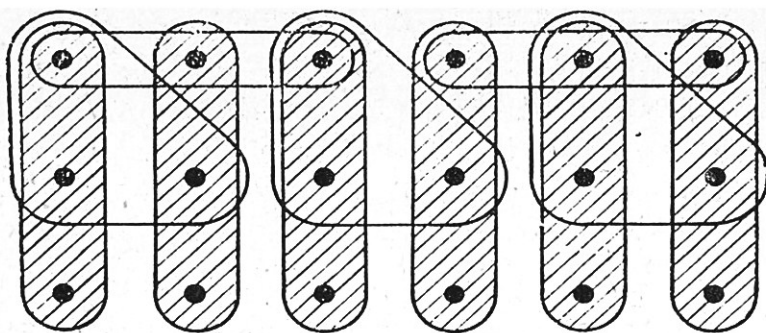
1.16. Tegyük föl, hogy kiválasztható volna $l < m - k$ halmaz, mondjuk $E_{j_1}, \dots, E_{j_l}$ úgy, hogy ezek S minden pontját lefedik. Válasszunk ki közülük maximális számú diszjunktat, legyenek ezek mondjuk $E_{j_1}, \dots, E_{j_p}$. Ezek lefednek $2p$ pontot. A maradék $m - 2p$ pontot más-más E_{j_i} halmaz kell hogy lefedje, ezért

$$l \geq p + (m - 2p) = m - p \geq m - k,$$

ellentmondás.

1.17. A 22. ábrán látható halmazrendszerben a 6 „függőleges” halmaz mindent lefed. A „mohó” algoritmussal (ha a lehetőségek között nem választunk szerencsésen) azonban lehetséges, hogy először a két „vízszintes” halmazt, majd a három „háromszöget”, végül a hat „függőleges” halmazt választjuk ki.

1.18. Tegyük fel, hogy a „mohó” algoritmus a lépésen át diszjunkt, vagyis 4 új elemet lefedő, b lépésen át 3 új elemet,



22. ábra

c lépésen át 2 új elemet, d lépésen át 1 új elemet lefedő halmazt ad. Ekkor

$$m = 4a + 3b + 2c + d \geq 4k_0$$

(ahol k_0 ismét a lefedő halmazok minimális száma). A szövegbeli bizonyításhoz hasonlóan adódik, hogy

$$3b + 2c + d \geq 3k_0,$$

$$2c + d \geq 2k_0,$$

$$d \geq k_0.$$

E négy egyenlőtlenséget rendre $3/12$, $1/12$, $2/12$, $6/12$ -del szorozva és összeadva az adódik, hogy

$$a + b + c + d \geq \frac{25}{12}k_0.$$

1.19. Mivel egy visszamaradó halmaz r pontot tartalmaz, és a többi is tartalmaz legalább egy-egy olyat, mely a többiben nincsen benne (különben elhagyható volna a rendszerből),

$$|S| \geq r + k - 1.$$

Másrészt nyilván

$$|S| \leq rk_0,$$

amiből az állítás következik.

1.20. Az előzőek mintájára egy játékká fogalmazhatjuk át a feladatot: Barátomnak most azt kell megmondania, van-e kör a gráfban. Nyerő stratégiám: csak akkor mondok nemet, ha ellenkező esetben az igenlően megválaszolt élek kört alkotnának.

Így barátom csak úgy nyerhet, ha $\binom{n}{2} - 1$ lépés után még az utolsónak maradt (u, v) él sem alkotna kört a többivel, vagyis u -t v -vel nem kötné össze igenlően megválaszolt út. Legyen U az u -ból igenlően megválaszolt éleken át elérhető pontok halmaza, V pedig a többi pont. Ekkor $u \in U$, $v \in V$, és az U -t V -vel összekötő élek

(u, v) kivételével nemlegesen lettek megválaszolva. De ezek bármelyike olyan, hogy az igenlően megválaszolt élekkel együtt nem alkotott volna kört, ami ellentmondás.

1.21. Legyen $k = \lfloor \log_2 n \rfloor$, vagyis $2^k \leq n < 2^{k+1}$. Tegyük fel először, hogy $2^k < n$. Ekkor $k+1$ forduló a kieséses verseny. Elérhetjük azt, hogy mindenki csak egyetlen fordulóban legyen erőnyerő (vagyis a versenyzők páratlan száma miatt játék nélkül továbbjutó versenyző) és azt is, hogy minden forduló után legfeljebb két korábbi erőnyerő maradjon versenyben. Ugyanis, ha erre a k -edik fordulóig vigyázni tudtunk, akkor a k -edik fordulóban a következőt tehetjük: ha két korábbi erőnyerő van, ezeket egymással sorsoljuk össze; ha csak egy van, akkor ezt is összesorsoljuk valakivel. Mivel a fordulóban csak egy új erőnyerő lesz, kikötéseinkhez így továbbra is tartani tudtuk magunkat.

Így a győztes legalább k mérkőzést játszott, mire megnyerte a versenyt, tehát csak $n - k - 1$ további mérkőzésre van szükség, hogy eldöntsük, abszolút legjobb-e. Így összesen $(n-1) + (n-k-1) = 2n - k - 2$ mérkőzés kellett.

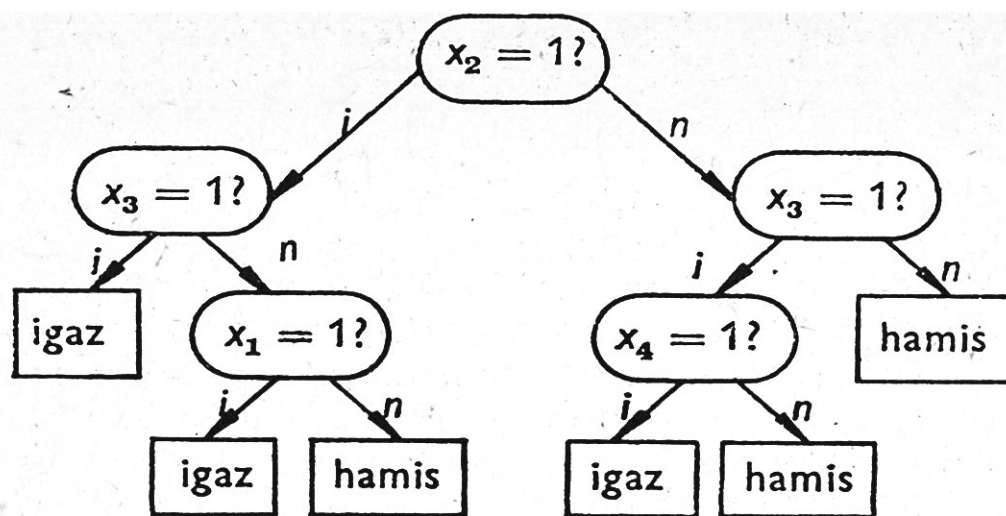
Ha $n = 2^k$, akkor csak k forduló van, de a győztes mindegyikben játszik, így az eredmény ugyanaz.

1.22. Először is vegyük észre, hogy mindig marad veretlen versenyző, vagyis végül lesz „abszolút” győztes.

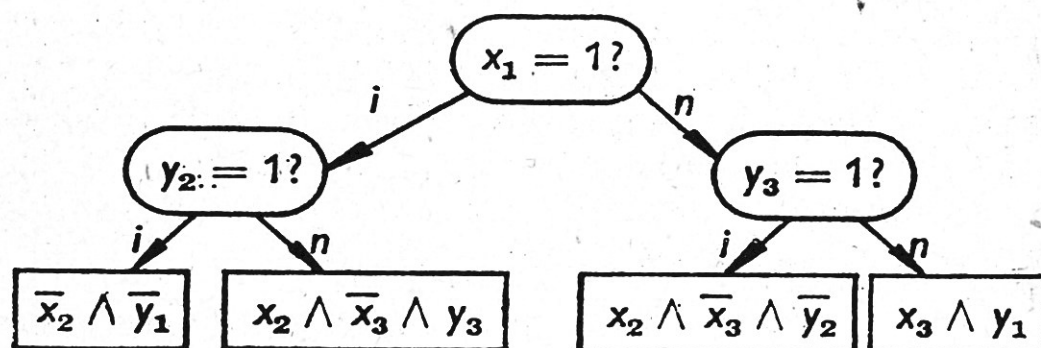
Nevezzük „valódinak” azokat a játszmákat, melyeket veretlen versenyzők játszottak egymással. Ezek száma nyilván $n-1$. k szerinti teljes indukcióval következik, hogy mikorra egy versenyző k ilyen játszmát megnyer, ő és az általa legyőzöttek legalább $2^k - 1$ valódi játszmát le kellett hogy játsszanak. Mivel a valódi játszmák száma $n-1$, a győztes legfeljebb $\lfloor \log_2 n \rfloor$ valódi játszmát játszott. Mivel mindenkit megvert, legalább még $n-1 - \lfloor \log_2 n \rfloor$ „nem valódi” játszmát kellett játszania. Ez az $n-1$ valódi játszmával együtt már $2n - 2 - \lfloor \log_2 n \rfloor$ játszma.

1.23. Számoljuk össze ismét, hogy a $v_2, \dots, v_n$ pontok által feszített G' gráfhoz hányféleképpen lehet v_1 -et kötni úgy, hogy ne kapjunk teljes pontot. Ha G' -nek $k \geq 1$ teljes pontja van, akkor v_1 -et G' nem teljes pontjai közül tetszőlegesen köthetjük össze, ez 2^{n-k} lehetőség. Ez páros, kivéve azt az egyetlenegy esetet, amikor G' teljes gráf. A $k=0$ esetben v_1 -et G' tetszőleges pontjaival köthetjük össze, de nem minddel. Ez $2^{n-1} - 1$ lehetőség. Így 1-gyel több esetben kapunk páratlan számot, mint ahány teljes pont nélküli G' van. Más szóval a teljes pont nélküli n , ill. $n-1$ pontú gráfok száma ellentétes párosságú. Ebből következik, hogy a teljes pontot nem tartalmazó n pontú gráfok száma $(n+1)$ -gyel azonos párosságú.

1.24. Lásd a 23. ábrát!



23. ábra



24. ábra

1.25. A 24. ábrán az alul bekeretezett függvények már legfeljebb 3–3 változót tartalmaznak, így értékük legfeljebb 3 további kérdéssel meghatározható.

2. fejezet

2.1. Az $A_1, \dots, A_{2n+1}$ állításoknak akkor és csak akkor igaz több, mint fele, ha bármely $n+1$ között van igaz. Így mindenféleképpen képezve $n+1$ diszjunkcióját és ezek konjunkcióját, a kívánt normálformát kapjuk.

Megmutatjuk, hogy $\binom{2n+1}{n}$ -nél kisebb fokú normálforma nem jelentheti ugyanezt. Tekintsünk ugyanis egy tetszőleges normálformát, amely azt fejezi ki, hogy $A_1, \dots, A_{2n+1}$ közül több, mint a fele igaz. Megmutatjuk, hogy bárhogyan is választunk ki az A_i -k közül $n+1$ darabot (mondjuk $A_1, \dots, A_{n+1}$ -et), lesz olyan tényező, melyben pontosan ezek szerepelnek negálatlanul; ebből az állítás nyilvánvaló. Válasszuk először is $A_1, \dots, A_{n+1}$ értékét

hamisnak, a többi változóét igaznak. Ekkor a normálforma értéke hamis kell, hogy legyen; van tehát olyan tényező, amelyik hamis, vagyis melyben $A_{n+2}, \dots, A_{2n+1}$ biztosan nem szerepelnek negátlanul.

Megmutatjuk, hogy $A_1, \dots, A_{n+1}$ mindegyike viszont kell hogy szerepeljen, és pedig negátlanul. Tegyük fel ugyanis, hogy pl. A_{n+1} nem, vagy csak negátlan szerepel ebben a tényezőben. Válasszuk $A_1, \dots, A_n$ értékét hamisnak, a többi igaznak, ekkor a tekintett tényező, és így az egész normálforma értéke hamis, ellentmondás.

2.2. Minden körülkerített halmazhoz, valamint minden $(x_i, \bar{x}_i)$ párhoz rendeljünk egy pontot. Minden x_i ponthoz konstruáljuk meg új pontok egy halmazát, mely az $(x_i, \bar{x}_i)$ párnak, és az x_i -t tartalmazó „körülkerített” halmazoknak megfelelő új pontokból áll. Tegyük meg ugyanezt az $\bar{x}_i$ pontokkal is. Ekkor a feladat: lefedhetők-e az új pontok 5 ilyen halmazzal?

2.3. Feltéhetjük, hogy a gráf összefüggő, mert az eljárást komponensenként alkalmazhatjuk. Egy pontot színezzünk pirosra, szomszédait kékre, azok szomszédait pirosra s. i. t. Ha van egyáltalán jó színezés, az így kapott színezés az.

2.4. Legyen G egy gráf. Vegyünk fel $k-3$ pontot, és kössük össze őket egymással és G pontjaival. A nyert G' gráf akkor és csak akkor színezhető ki k színnel, ha G kiszínezhető három színnel. Így a 3 színnel való színezhetőség problémáját visszavezettük a k színnel való színezhetőségre.

2.5. Első sor: izomorfak (a hiányzó élek mindkettőben 7-szöget alkotnak).

Második sor: izomorfak (a bal oldali rajz nyilván egy kocka élhálójá; egymással átellenes csúcsait rajzoltuk egymás fölé a jobb oldalon).

Harmadik sor: nem izomorfak (a bal oldali ábrán van 4-szög, a jobb oldalin nincsen).

2.6. Ha a kismadár fülünkbe csicsergi, melyik pontot melyik pontnak feleltessük meg, könnyű ellenőrizni, hogy igaza van-e.

3. fejezet

3.1. k szerinti teljes indukcióval bizonyítunk. $k=0$ esetén (2) a $T(n) \leq T(n)$ egyenlőtlenségbe megy át. Tegyük fel, hogy (2) igaz minden k -nál kisebb számra. Bebizonyítjuk, hogy ekkor igaz k -ra is. Alkalmazzuk (1)-et:

$$T(2 \cdot 2^{k-1}n) \leq 3T(2^{k-1}n) + 9 \cdot 2^{k-1}n.$$

Alkalmazzuk az indukciós feltevést:

$$T(2^{k-1}n) \leq 3^{k-1}T(n) + 9n(3^{k-1} - 2^{k-1}).$$

Helyettesítsünk be az előbbi egyenlőtlenségbe:

$$\begin{aligned} T(2 \cdot 2^{k-1}n) &\leq 3 \cdot 3^{k-1}T(n) + 9n \cdot 3 \cdot (3^{k-1} - 2^{k-1}) + 9 \cdot 2^{k-1}n = \\ &= 3^k T(n) + 9n(3^k - 2^k). \end{aligned}$$

Tehát (2) igaz k -ra is, ami azt jelenti, hogy (2) általánosan igaz.

3.2. E feladatot használjuk fel arra, hogy blokkdiagramot adjunk *Karacuba* módszerére. Mivel blokkdiagrammal közvetlenül nem lehet átírni a *Karacubához* hasonló, *rekurziót* tartalmazó algoritmusokat, egy *fogásra* lesz szükségünk, amely az e fejezetben található többi rekurzív algoritmus esetében is alkalmazható. A fogás abban áll, hogy a részeredmények olyan $X[p]$, $X[p]$, $Z[p]$ *indexes változók* lesznek, melyeknek indexei nem számok, hanem *véges sorozatok*. A $p = p_1, p_2, \dots, p_k$ sorozat elemei a 0, 1, 2 számok lehetnek. A p sorozat hosszát, k -t, $l(p)$ -vel jelöljük. Jelen esetben, amikor 16-jegyű számok szorzásáról van szó, $l(p)$ legfeljebb 4 lesz. Egy $p = p_1 p_2 \dots p_k$ sorozat esetén p' -vel jelöljük a $p_2 \dots p_k$ sorozatot. Kényelmes bevezetni a 0 hosszúságú sorozatot, melyet λ -val jelölünk.

Ha $u = \pm \overline{u_{2n-1} \dots u_0}$ egy $2n$ jegyű szám, akkor, mint előbb, U_1 -gyel és U_0 -lal jelöljük u „bal”-, illetve „jobb”-felét. Ha u negatív, akkor U_0 és U_1 is negatív.

Bemenő adatok az $a = \overline{a_{15} \dots a_0}$, $b = \overline{b_{15} \dots b_0}$ 2-es számrendszerbeli számok. Az algoritmus befejezésekor $X(\lambda) = ab$. Ezt úgy bizonyíthatjuk, hogy az $X[p] = Y[p] \cdot Z[p]$ azonosságot bebizonyítjuk sorban az $l(p) = 4, 3, 2, 1, 0$ esetekre (tehát $4 - l(p)$ szerinti teljes indukcióval).

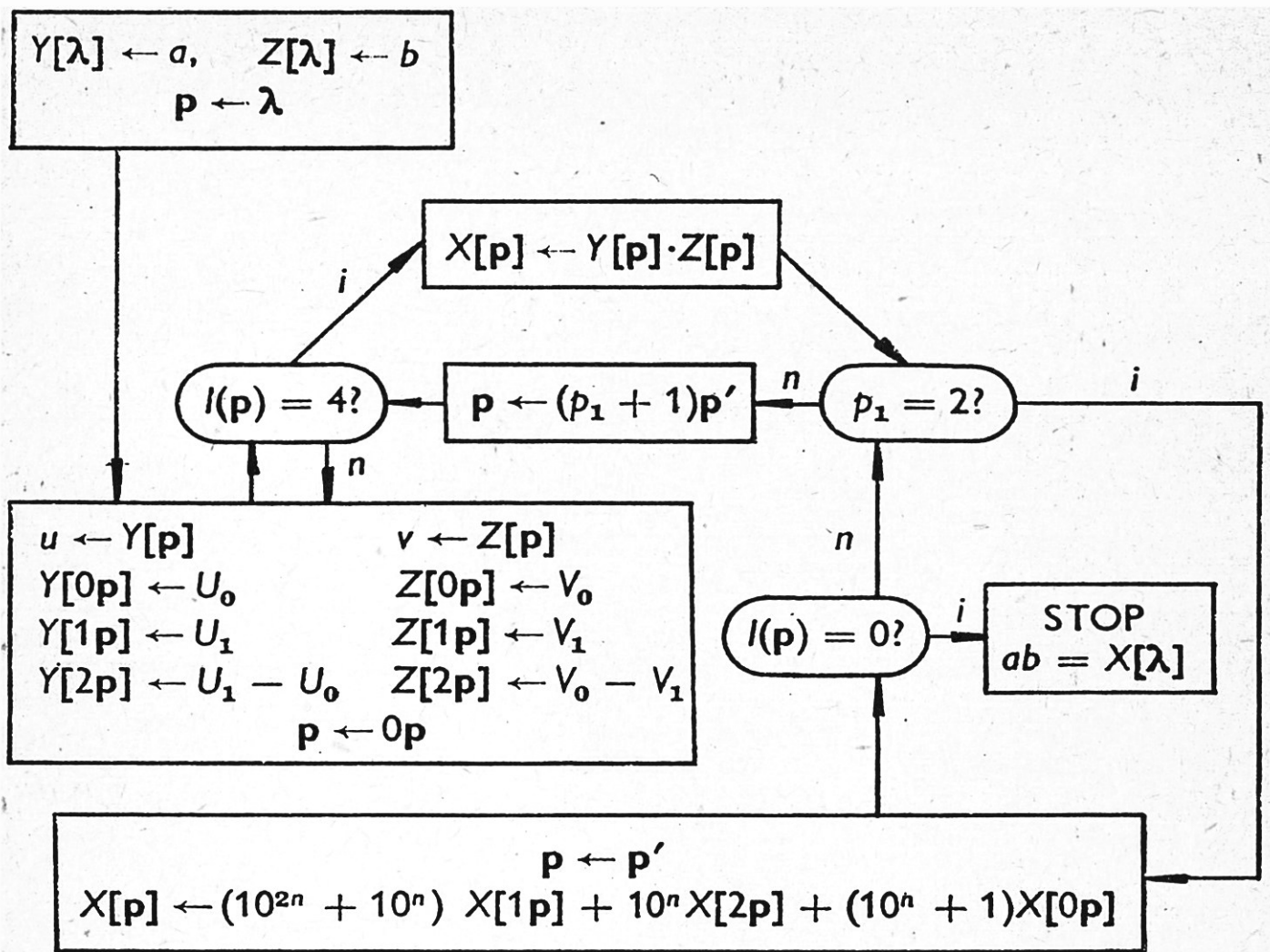
3.3. Bizonyítsunk teljes indukcióval, akárcsak a 3.1. feladatban.

3.4. Bizonyítsunk teljes indukcióval, akárcsak a 3.1. feladat megoldásában.

3.5. Adottak az $R(0), R(1), \dots, R(2n-2)$ számok, keresendők az R polinom együtthatói. Keressük $R(x)$ -et az

$$R(x) = d_0 + d_1 x + d_2 \frac{x(x-1)}{2} + \dots + d_{2n-2} \binom{x}{2n-2}$$

alakban. Legyen $\Delta^0 R(x) = R(x)$, $\Delta^1 R(x) = R(x+1) - R(x)$, $\dots$, $\Delta^{k+1} R(x) = \Delta^k R(x+1) - \Delta^k R(x)$. Egyszerű helyettesítés mu-



tatja, hogy $\Delta^k R(0) = d_k$. Így tehát d_k megtalálható, ha kiszámoljuk a

$$\begin{aligned}
 d_0 &= R(0) & R(1) & R(2) & R(3) \dots \\
 d_1 &= \Delta R(0) & \Delta R(1) & \Delta R(2) \dots \\
 d_2 &= \Delta^2 R(0) & \Delta^2 R(1) \dots \\
 d_3 &= \Delta^3 R(0) \dots
 \end{aligned}$$

háromszög alakú táblázatot, ahol a táblázat minden eleme a felette álló két elem *különbsége*. Ez $\frac{1}{2}(2n-1)^2$ darab kivonást jelent. Az $R(x)$ polinom együtthatóit ezután további $(2n-1)^2$ lépésben kiszámolhatjuk az

$$R(x) = d_0 + x \left\{ d_1 + \frac{x-1}{2} \left[d_2 + \frac{x-2}{3} (d_3 + \dots) \right] \right\}$$

képlet alapján.

3.6. A jobb oldali összeg

$$\frac{1}{r} \sum_{k=0}^{r-1} \hat{a}_k \varepsilon^{-jk} = \frac{1}{r} \sum_{k=0}^{r-1} \varepsilon^{-jk} \sum_{l=0}^{n-1} a_l \varepsilon^{lk} = \frac{1}{r} \sum_{l=0}^{n-1} a_l \sum_{k=0}^{r-1} \varepsilon^{k(l-j)}.$$

$\varepsilon^{l-j} = \delta$ ugyancsak egy r -edik egységgyök lesz. Ha $l=j$, akkor $\delta=1$. Ha $l \neq j$, akkor $\sum_k \delta^k = 0$. Ezért a fenti összeg egyenlő a_j -vel.

3.7. Láttuk, hogy

$$\frac{x^3+1}{2x^2-x} = \frac{1}{2}x + \frac{1}{4} + \frac{\frac{1}{4}x+1}{2x^2-x}.$$

Folytatva az osztást,

$$\left(\frac{1}{4}x+1\right):(2x^2-x) = \frac{1}{8}x^{-1} + \frac{9}{16}x^{-2} + \frac{9}{32}x^{-3} + \dots$$

$$\frac{9}{8}$$

$$\frac{9}{16}x^{-1}$$

$$\frac{9}{32}x^{-2}$$

3.8. Milyen feltételeknek kell teljesülnie c_i -re? Írjuk fel a

$$\sum_{i=0}^m a_i x^i = \left(\sum_{i=0}^n b_i x^i \right) \left(\sum_{i=-\infty}^{m-n} c_i x^i \right)$$

egyenlőséget, szorozzuk be a jobb oldalon, és írjuk fel az egyenleteket, amelyek biztosítják, hogy x azonos kitevőjű hatványaihoz a jobb és a bal oldalon azonos együtthatók tartoznak:

$$a_m = b_n c_{m-n}, \quad a_{m-1} = b_m c_{m-n-1} + b_{n-1} c_{m-n}, \quad \dots$$

Ezekből c_i -t kifejezve, éppen azokat a műveleteket kell elvégeznünk, amelyeket a polinomosztás követel.

3.9. Legyen a_{i-1} , a_i , a_{i+1} a sorozat három tagja, $1 \leq i \leq k$. Az $a_{i-1} = q_i a_i + a_{i+1}$ egyenlőségből látható, hogy egy p szám akkor és csak akkor közös osztója az a_{i-1} és a_i számoknak, ha az a_i és a_{i+1} számoknak is, azaz $(a_{i-1}, a_i) = (a_i, a_{i+1})$. Innen következik, hogy $(a_0, a_1) = (a_1, a_2) = \dots = (a_k, a_{k+1}) = a_k$.

3.10. Definíció szerint

$$\bar{Q}_i = (-1)^{i-1} Q_i, \quad \bar{F}_i = (-1)^{\left[\frac{i}{2}\right]} \cdot F_i.$$

(9) ekkor így írható:

$$(-1)^{\left[\frac{i-1}{2}\right]} F_{i-1} = (-1)^{\left[\frac{i}{2}\right]} F_i \cdot (-1)^{i-1} Q_i - (-1)^{\left[\frac{i+1}{2}\right]} F_{i+1}.$$

Vegyük észre, hogy

$$(-1)^{\left[\frac{i}{2}\right]+i-1} = (-1)^{\left[\frac{i-1}{2}\right]} = (-1)^{\left[\frac{i+1}{2}\right]+1}.$$

Ezért a fenti egyenlőség átmegy az alábbiába:

$$F_{i-1} = F_i Q_i + F_{i+1}.$$

3.11. (Útmutatás) Legyen

$$F_0(x) = u_0 x^m + u_1 x^{m-1} + \dots + u_m,$$

$$F_1(x) = v_0 x^n + v_1 x^{n-1} + \dots + v_n,$$

$$F_0^{(t)}(x) = u_0 x^t + u_1 x^{t-1} + \dots + u_t, \quad t > 0 \text{ egész szám,}$$

$$F_1^{(t)}(x) = x^{m-n}(v_0 x^t + \dots + v_t).$$

Legyen $Q_1^{(t)}, Q_2^{(t)}, \dots$ az $F_0^{(t)}$ és $F_1^{(t)}$ -ből euklidészi algoritmussal kapott hányadossorozat, $n_1^{(t)}, n_2^{(t)}, \dots$ a $Q_1^{(t)}, Q_2^{(t)}, \dots$ fokszámai, $l_j = k_j + n_{j+1}$. Mutassuk meg j szerinti teljes indukcióval, hogy

1. ha $k_j \leq t$, akkor $Q_i = Q_i^{(t)}$ ($i = 1, \dots, j$);
2. ha $l_j \leq t$, akkor $n_i = n_i^{(t)}$ ($i = 1, \dots, j+1$)

és $F_{j+1}^{(t)}$ első $t - l_j$ darab tagja megegyezik F_{j+1} első $t - l_j$ darab tagjával.

3.12. Mint tudjuk,

$$L(2m) \leq 2L(m) + 4K(m) + 2K(2m) + 2m^2. \quad (*)$$

Ismeretes a 3.1.2. pontból, hogy van egy olyan c konstans, amellyel $K(m) \leq cn^\alpha$, ahol $\alpha = \log_2 7$. Bebizonyítjuk, teljes indukcióval, hogy

$$L(n) \leq (18c + 4)n^\alpha. \quad (**)$$

$n=1$ esetén $L(n)=1$. $n=2$ esetén $(*)$ -ből, figyelembe véve, hogy $K(2) \leq 7$, azt kapjuk, hogy

$$L(2) \leq 2 + 4 + 14 + 8 < 18 \cdot 7 + 4.$$

Tegyük fel, hogy $(**)$ igaz már minden n -nél kisebb számra, és $n \geq 3$. Ha n páros ($n=2m$), akkor $(*)$ alkalmazható. Ha n páratlan, akkor készítsük el azt az $(n+1) \times (n+1)$ -es $\tilde{A}$ mátrixot, melyre $\tilde{a}_{ij} = a_{ij}$ ($i, j = 1, \dots, n$), $\tilde{a}_{(n+1)(n+1)} = 1$, minden más elem 0. Akkor $\tilde{A}^{-1}$ bal felső sarkában A inverze áll, ezért A helyett elég $\tilde{A}$ -ot invertálni. Tehát minden n -re igaz az

$$L(n) \leq 2L\left(\left\lceil \frac{n}{2} \right\rceil\right) + 4K\left(\left\lceil \frac{n}{2} \right\rceil\right) + 2K(n+1) + 2\left(\frac{n+1}{2}\right)^2$$

egyenlőtlenség. Alkalmazzuk az indukciós feltevést (mint tudjuk, $2^z = 7$):

$$L(n) \leq 2 \cdot (18c + 4) \left(\frac{n+1}{2}\right)^\alpha + 4c \left(\frac{n+1}{2}\right)^\alpha + 2c(n+1)^\alpha + 2 \left(\frac{n+1}{2}\right)^2 < \\ < (n+1)^\alpha \left[c \left(\frac{36}{7} + \frac{4}{7} + 2 \right) + \frac{8}{7} + \frac{1}{2} \right] = n^\alpha \left(1 + \frac{1}{n} \right)^\alpha \left(\frac{54}{7}c + \frac{23}{14} \right).$$

Mivel

$$\left(1 + \frac{1}{n} \right)^\alpha \leq \left(\frac{4}{3} \right)^\alpha < \frac{7}{3},$$

ezért

$$L(n) < n^\alpha \cdot \frac{7}{3} \left(\frac{54}{7}c + \frac{23}{14} \right) < (18c + 4)n^\alpha.$$

3.13. $x_1^2 + x_2^2 = (x_1 + ix_2)(x_1 - ix_2)$, két szorzás helyett csak egyet kellett elvégezni. A valós számokra vonatkozó bizonyítás kihasználta, hogy $u_1^2 + \dots + u_n^2 = 0$ csak akkor állhat fenn, ha $u_1, \dots, u_n$ mindegyike 0. Ez nem igaz komplex számokra.

3.14. Ha ezt az állítást közvetlenül akarnánk bizonyítani, akkor a 3.3.1. pontban szereplő első tételt kellene úgy általánosítani, hogy y helyett az y vektor szerepeljen, és az algoritmus végeredménye ne csak szám, hanem vektor (esetünkben az f vektor) lehessen. Ez nem nehéz, mi azonban — *Strassen* nyomán — egy egyszerűbb, közvetett utat választunk: *Pan* tételét használjuk fel. Tegyük fel, hogy az $\mathbf{Xy} = \mathbf{f}$ vektor r lépésben kiszámolható. Legyen speciálisan $\mathbf{y} = (y, y^2, \dots, y^n)$. Ekkor (az x_{ij} -vel végzett nem-lineáris műveleteket számolva) további $(k-1)$ lépésben kiszámolható az

$$(x_{10} + f_1) + (x_{20} + f_2)y^{n+1} + (x_{30} + f_3)y^{2(n+1)} + \dots = \\ = \sum_{i=0}^k \sum_{j=1}^n x_{ij} y^{(i-1)(n+1)+j}$$

$(nk + k - 1)$ -ed fokú polinom is, ahol $x_{10}, x_{20}, \dots, x_{k0}$ új változók. Márpedig *Pan* tétele szerint ehhez legalább $nk + k - 1$ művelet kell. Ezért $r + k - 1 \geq nk + k - 1$, $r \geq nk$.

3.15. Vegyük észre, hogy a különbség (2) és (3) között csak annyi, hogy (3) a második nevezőbeli

$$\frac{y(x+y)}{3-yx} = yz_4$$

kiszámításához felhasználja z_4 értékét, (2) pedig nem.

3.16. Alkalmazzunk teljes indukciót a műveletek L számára. $L=0$ esetén igaz az állítás. Tegyük fel, hogy igaz az állítás minden olyan F' kifejezésre, mely L -nél kevesebb műveletet tartalmaz. Ha az F kifejezés L műveletet tartalmaz, akkor $F = F_1 \pm F_2$, $F_1 F_2$, F_1/F_2 vagy F_2/F_1 , ahol F_2 nem tartalmazza x_1 -et. F_1 már L -nél kevesebb műveletet tartalmaz, ezért a kívánt alakú. Egyszerű számolás mutatja, hogy ekkor F is a kívánt alakú lesz.

3.17. Legyen z_i egy maximális magasságú olyan pont, melyre (a) teljesül. Akkor (b) is teljesül, mert különben z_j is eleget tenne (a)-nak, és magasabban lenne, mint z_i .

3.18. Megmutatjuk, hogy ha egy f formulában a műveletek száma legfeljebb L , akkor F kiszámolható $1 + c \log L$ szakaszban, ahol

$$c = \frac{4}{\log(3/2)}.$$

L szerinti teljes indukciót alkalmazunk. $L=1$ -re az állítás nyilván teljesül. Tegyük fel, hogy igaz minden, L -nél kisebb művelet-száma. Legyen F egy L műveletet tartalmazó formula. Akkor diagramjában a 2. feladat szerint van egy olyan z_i pont, hogy $F = F'(z_i, x_1, \dots, x_n)$, ahol F' egy legfeljebb $L/3$ műveletet tartalmazó formula, és z_i -re (b) is teljesül. Mint a 3.1.6. feladatból tudjuk,

$$F' = \frac{z_i G_1 + G_2}{z_i G_3 + G_4},$$

ahol $G_1, \dots, G_4$ nem tartalmazza z_i -t és legfeljebb $2L/3$ műveletet kíván. Legyenek z_j és z_k a z_i felett álló pontok. Kiszámításuk (b) szerint legfeljebb $2L/3$ műveletet kíván. Ezért $G_1, \dots, G_4, z_j$ és z_k párhuzamosan számolható, az indukciós feltevés szerint $1 + c \log(2L/3)$ számú szakaszban. Innen, mint látható, $F = F'$ további 4 szakaszban kiszámolható, a szakaszok száma tehát legfeljebb

$$1 + c \log L + c \log(2/3) + 4 = 1 + c \log L.$$

3.19. Első lehetőség: járjunk el úgy, mint az előző három feladatban, csak az

$$F = \frac{G_1 x + G_2}{G_3 x + G_4}$$

alak helyett használjuk az

$$F = (x \wedge G_1) \vee (\neg x \wedge G_2)$$

alakot.

Másik lehetőség: fejezzük ki a logikai műveleteket az alapműveletekkel. Legyen 0 a hamis, 1 az igaz érték megfelelője. Ekkor

$$\neg x = 1 - x, \quad x \wedge y = xy, \quad x \vee y = x + y - xy.$$

Ezután alkalmazzuk *Brent* tételét!

4. fejezet

4.1. Vegyük sorra a számokat 2-től $\lceil \sqrt{x} \rceil$ -ig, és mindegyikkel osszuk el x -et. Ha x valamelyikkel osztható, álljunk meg és írjuk ki: $f(x) = 1$. Ellenkező esetben, $\lceil \sqrt{x} \rceil$ -szel való osztás után, írjuk ki: $f(x) = 0$.

4.2. Vegyünk két gépet, T_1 -et és T_2 -t, melyek közül T_2 kétszer lassabban dolgozik, mint T_1 : amíg T_1 két lépést hajt végre, addig T_2 egyet. Indítsuk egyszerre T_1 -et és T_2 -t ugyanazzal a P programmal és x bemenő adatokkal, és T_2 minden (azaz T_1 minden második) lépése után ellenőrizzük, nincs-e a két gép *azonos állapotban*? T_1 és T_2 azonos állapotban van, ha

1. minden i -re az i -edik regiszter tartalma ugyanaz a T_1 gépen, mint T_2 -n,

2. T_1 és T_2 a P program ugyanazon utasításának végrehajtásánál tart.

(Vegyük észre, hogy az 1. feltétel ellenőrzéséhez nem kell végtelen sok regisztert végignézni, csak annyit, amennyi eddig már felhasználásra került.)

A P program az x adatok mellett akkor és csak akkor vezet ciklushoz, ha valamelyik lépésnél T_1 és T_2 azonos állapotba kerül.

4.3. Mint a megállási feladat definíciója előtti apróbetűs részből következik, van egy olyan P_T program, hogy a T gép a P_T programmal ugyanazt számolja ki a $\bar{Q} + x$ sorozatból, mint a Q programmal az x -ből. Ezért a megállási feladat így is fogalmazható: Döntsük el, hogy a T gép a (rögzített) P_T program, és az y adatszalag mellett megáll-e? Ez a feladat most csak az y sorozat tulajdonságának eldöntését kívánja. Ha megoldható volna, akkor az eldöntő eljárás azokban az esetekben is működne, amikor $y = \bar{Q} + x$.

4.4. Legyen $S_1, S_2, \dots$ a négyzeteknek egy korlátlanul növekvő sorozata, amelyben S_i része S_{i+1} -nek. Ha minden S_i -nek van egy lefedése, akkor S_1 -nek van olyan L_1 lefedése is, amely minden $i > 1$ -re kiterjeszthető S_i lefedésévé. Ez abból következik, hogy S_i bármely lefedése S_1 -nek is szolgáltatja egy lefedését, és S_1 -nek csak véges számú különböző lefedése lehet. Ezek valame-

lyike tehát végtelen sok (és ezért minden) S_i lefedésévé kiterjeszthető. Továbbmenve, van S_2 -nek olyan L_2 lefedése is, mely L_1 kiterjesztése, és minden $i > 2$ -re kiterjeszthető S_i lefedésévé stb. Az így kapott $L_1, L_2, \dots$ sorozat a sík lefedését adja.

4.5. Tegyük fel, hogy minden készlet vagy olyan, hogy vele a síkot nem lehet lefedni, vagy olyan, hogy vele a sík periodikusan lefedhető.

Megmutatjuk, hogy ekkor a dominóproblémára volna megoldó algoritmus. Ha a K készlettel a síkot nem lehet befedni, akkor (az előző feladat miatt) előbb-utóbb található olyan nagy négyzet, amelyet nem lehet lefedni. Ha a síkot periodikusan le lehet fedni, akkor viszont előbb-utóbb találunk egy nagy négyzetet, ami úgy fedhető le, hogy ez a lefedés periodikusan folytatható. Ezért, ha más eset nem volna lehetséges, akkor elég állhatatos keresés bármely készlet esetén eldönthetné, hogy a két eset közül melyik áll fenn.

4.6. A fenti bizonyítást csak annyiban kell módosítani, hogy máshogyan szabályozzuk, hogy mikor térjen vissza gépünk a $P(m)$ algoritmusra. Ehhez azt az egyszerű számelméleti tényt használjuk fel, hogy minden t természetes szám egyértelműen felbontható egy négyzetszám és egy négyzetmentes szám szorzatára (négyzetmentesnek nevezzük az olyan számot, mely nem osztható 1-től különböző természetes szám négyzetével). Szabályunk mármost az, hogy a gép a t -edik lépésben $P(m)$ működését szimulálja, ha t fenti felbontásában a négyzetszám-tényező m^2 , vagyis ha t egy négyzetmentes szám m^2 -szerese. Egyszerű számelméleti megfontolásokkal megmutatható, hogy N -ig legalább $N/4m^2$ ilyen szám van, amiből az állítás következik. Hasonló konstrukció volna elvégezhető 2^m ill. m^2 helyett minden olyan $f(m)$ függvényre, melyre $\sum 1/f(m)$ konvergens.

Tankönyvkiadó Vállalat. A kiadásért felelős: Petró András igazgató. 88-1138 — Dabasi Nyomda, Budapest — Dabas. Felelős vezető: Bálint Csaba igazgató. Raktári szám: 42 317. Felelős szerkesztő: Oláh Judit. Műszaki igazgatóhelyettes: Schröder Géza. Grafikai szerkesztő: Vara Tibor. Műszaki szerkesztő: Risai Klára. A kézirat nyomdába érkezett: 1988. május. Megjelent: 1989. február. Példányszám: 2000. Terjedelem: 11,25 (A/5) ív. Készült az 1987. évi második kiadás alapján, álló montírungról, íves ofszetnyomással, az MSZ 5601—59 és az MSZ 5602—55 szabvány szerint



42317